Second Edition

Data Networks

DIMITRI BERTSEKAS

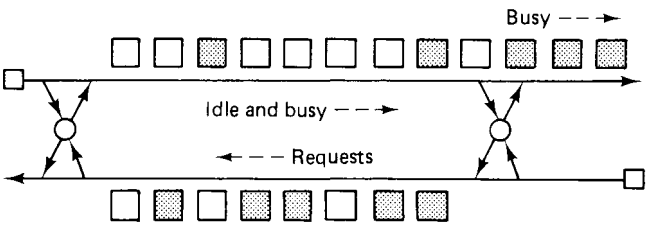
Massachusetts Institute of Technology

ROBERT GALLAGER

Massachusetts Institute of Technology

PRENTICE HALL, Englewood Cliffs, New Jersey 07632

4



Multiaccess Communication

4.1 INTRODUCTION

The subnetworks considered thus far have consisted of nodes joined by point-to-point communication links. Each such link might consist physically of a pair of twisted wires, a coaxial cable, an optical fiber, a microwave radio link, and so on. The implicit assumption about point-to-point links, however, is that the received signal on each link depends only on the transmitted signal and noise on that link.

There are many widely used communication media, such as satellite systems, radio broadcast, multidrop telephone lines, and multitap bus systems, for which the received signal at one node depends on the transmitted signal at two or more other nodes. Typically, such a received signal is the sum of attenuated transmitted signals from a set of other nodes, corrupted by distortion, delay, and noise. Such media, called *multiaccess media*, form the basis for local area networks (LANs), metropolitan area networks (MANs), satellite networks, and radio networks.

The layering discussed in Chapters 1 and 2 is not quite appropriate for multiaccess media. One needs an additional sublayer, often called the *medium access control* (MAC) sublayer, beween the data link control (DLC) layer and the modem or physical layer. The

purpose of this extra sublayer is to allocate the multiaccess medium among the various nodes. As we study this allocation issue, we shall see that the separation of functions between layers is not as clean as it is with point-to-point links. For example, feedback about transmission errors is part of the ARQ function of the DLC layer, but is often also central to the problem of allocation and thus flow control. Similarly, much of the function of routing is automatically implemented by the broadcast nature of multiaccess channels.

Conceptually, we can view multiaccess communication in queueing terms. Each node has a queue of packets to be transmitted and the multiaccess channel is a common server. Ideally, the server should view all the waiting packets as one combined queue to be served by the appropriate queueing discipline. Unfortunately, the server does not know which nodes contain packets; similarly, nodes are unaware of packets at other nodes. Thus, the interesting part of the problem is that knowledge about the state of the queue is distributed.

There are two extremes among the many strategies that have been developed for this generic problem. One is the "free-for-all" approach in which nodes normally send new packets immediately, hoping for no interference from other nodes. The interesting question here is when and how packets are retransmitted when collisions (*i.e.*, interference) occur. The other extreme is the "perfectly scheduled" approach in which there is some order (round robin, for example) in which nodes receive reserved intervals for channel use. The interesting questions here are: (1) what determines the scheduling order (it could be dynamic), (2) how long can a reserved interval last, and (3) how are nodes informed of their turns?

Sections 4.2 and 4.3 explore the free-for-all approach in a simple idealized environment that allows us to focus on strategies for retransmitting collided packets. Successively more sophisticated algorithms are developed that reduce delay, increase available throughput, and maintain stable operation. In later sections, these algorithms are adapted to take advantage of special channel characteristics so as to reduce delay and increase throughput even further. The more casual reader can omit Sections 4.2.3 and 4.3.

Section 4.4 explores carrier sense multiple access (CSMA). Here the free-for-all approach is modified; a packet transmission is not allowed to start if the channel is sensed to be busy. We shall find that this set of strategies is a relatively straightforward extension of the ideas in Sections 4.2 and 4.3. The value of these strategies is critically dependent on the ratio of propagation delay to packet transmission time, a parameter called β. If $\beta \ll 1$, CSMA can decrease delay and increase throughput significantly over the techniques of Sections 4.2 and 4.3. The casual reader can omit Sections 4.4.2 and 4.4.4.

Section 4.5 deals with scheduling, or reserving, the channel in response to the dynamic requirements of the individual nodes. We start with satellite channels in Section 4.5.1; here the interesting feature is dealing with $\beta \gg 1$. Next, Sections 4.5.2 to 4.5.4 treat the major approaches to LANs and MANs. These approaches can be viewed as reservation systems and differ in whether the reservations are scheduled in a free-for-all manner or in a round-robin manner. LANs are usually designed for the assumption that β is small, and Section 4.5.5 explores systems with higher speed or greater geographical coverage for which β is large.

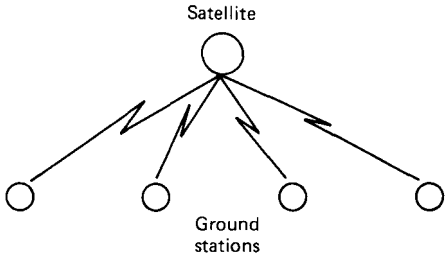
Finally, Section 4.6 explores packet radio. Here the interesting issue is that each node interferes only with a limited subset of other nodes; thus, multiple nodes can transmit simultaneously without interference.

Before going into any of the topics above in detail, we briefly discuss some of the most widely used multiaccess media.

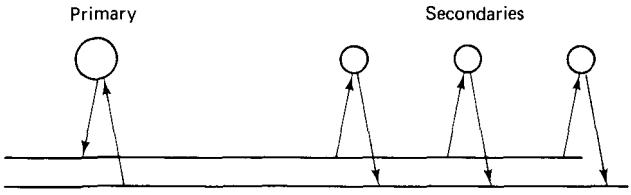
4.1.1 Satellite Channels

In a typical geosynchronous communication satellite system, many ground stations can transmit to a common satellite receiver, with the received messages being relayed to the ground stations (see Fig. 4.1). Such satellites often have separate antenna beams for different geographical areas, allowing independent reception and relaying between areas. Also, FDM (or TDM) can be used, permitting different earth stations within the region covered by a single antenna beam to be independently received.

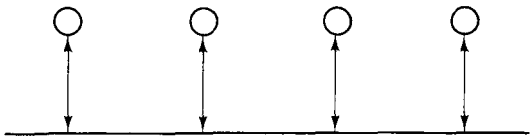
It is thus possible to use a satellite channel as a collection of virtual point-to-point links, with two virtual links being separated either by antenna beam or multiplexing. The potential difficulty with this approach is the same as the difficulty with using FDM or



(a) Satellite multiaccess channel



(b) Multidrop telephone line



(c) Multitap bus

Figure 4.1 Common multiaccess channels.

TDM for multiple sessions on a single point-to-point link, namely increased delay and underutilization of the medium.

In Section 4.5.1 we shall find that delay can be reduced and utilization increased by sharing the medium on a demand basis. This is more difficult than demand sharing (*i.e.*, statistical multiplexing) on a point-to-point link because the earth stations are not aware of the instantaneous traffic requirements of other earth stations. If several stations transmit at the same time (and in the same frequency band), their signals are garbled and incorrectly received.

4.1.2 Multidrop Telephone Lines

Another example of a multiaccess channel is a multidrop telephone line. Such lines connect one primary node with a number of secondary nodes; the signal transmitted by the primary node goes over one pair of wires and is received by all the secondary nodes. Similarly, there is a return pair of wires which carries the sum of the transmitted signals from all the secondary nodes to the primary node. Conceptually, this is like a satellite channel. The secondary nodes, or earth stations, share the path to the primary node, or satellite, whereas the primary node, or satellite, broadcasts to all the secondary nodes, or earth stations. Most communication on a multidrop phone line is intended to go from primary to secondary, or vice versa, whereas most communication on a satellite channel is relayed by the satellite from one earth station to another. Conceptually, this difference is not very important, since the major problem is that of sharing the channel from the secondary nodes to the primary, and it makes little difference whether the messages are removed at the primary node or broadcast back to all the secondary nodes.

The traditional mode of operation for multidrop telephone lines is for the primary node to poll (*i.e.*, request information from) each secondary node in some order. Each secondary node responds to its poll either by sending data back to the primary station or by indicating that it has no data to send. This strategy avoids interference between the secondary nodes, since nodes are polled one at a time, but there is a certain amount of inefficiency involved, both in the time to send a poll and the time to wait for a response from a node with no data.

4.1.3 Multitapped Bus

A third example of a multiaccess channel is a bus with multiple taps. In this case, each node can receive the signal sent by each other node, but again, if multiple nodes transmit at the same time, the received signal is garbled. We shall discuss this example later in the context of Ethernet, but for now, we observe that conceptually this channel is very similar to a satellite channel. Each node can communicate with each other node, but if nodes transmit simultaneously, the received signal cannot be correctly detected. The fact that nodes can hear each other directly here, as opposed to hearing each other via relay from the satellite, has some important practical consequences, but we will ignore this for now.

4.1.4 Packet Radio Networks

A fourth example of multiaccess communication is that of a packet radio network. Here each node is in reception range of some subset of other nodes. Thus, if only one node in the subset is transmitting, the given node can receive the transmission, whereas if multiple nodes are transmitting simultaneously in the same band, the reception will be garbled. Similarly, what one node transmits will be heard by a subset of the other nodes. In general, because of different noise conditions at the nodes, the subset of nodes from which a given node can receive is different from the subset to which the given node can transmit. The fact that each receiver hears a subset of transmitters rather than all transmitters makes packet radio far more complex than the other examples discussed. In the next four sections, we study multiaccess channels in which a receiver can hear all transmitters, and then in Section 4.6, we discuss packet radio networks in more detail.

4.2 SLOTTED MULTIACCESS AND THE ALOHA SYSTEM

Satellite channels, multidrop telephone lines, and multitap bus systems all share the feature of a set of nodes sharing a communication channel. If two or more nodes transmit simultaneously, the reception is garbled, and if none transmit, the channel is unused. The problem is somehow to coordinate the use of the channel so that exactly one node is transmitting for an appreciable fraction of the time. We start by looking at a highly idealized model. We shall see later that multiaccess channels can often be used in practice with much higher utilization than is possible in this idealized model, but we shall also see that these practical extensions can be understood more clearly in terms of our idealization.

4.2.1 Idealized Slotted Multiaccess Model

The idealized model to be developed allows us to focus on the problem of dealing with the contention that occurs when multiple nodes attempt to use the channel simultaneously. Conceptually, we view the system as in Fig. 4.1(a), with m transmitting nodes and one receiver.

It will be observed that aside from some questions of propagation delay, this same model can be applied with m nodes that can all hear each other (*i.e.*, the situation with a multitap bus). We first list the assumptions of the model and then discuss their implications.

1. *Slotted system.* Assume that all transmitted packets have the same length and that each packet requires one time unit (called a slot) for transmission. All transmitters are synchronized so that the reception of each packet starts at an integer time and ends before the next integer time.
2. *Poisson arrivals.* Assume that packets arrive for transmission at each of the m transmitting nodes according to independent Poisson processes. Let λ be the overall arrival rate to the system, and let λ/m be the arrival rate at each transmitting node.

3. *Collision or perfect reception.* Assume that if two or more nodes send a packet in a given time slot, then there is a *collision* and the receiver obtains no information about the contents or source of the transmitted packets. If just one node sends a packet in a given slot, the packet is correctly received.
4. *0,1,e Immediate feedback.* Assume that at the end of each slot, each node obtains feedback from the receiver specifying whether 0 packets, 1 packet, or more than one packet (e for error) were transmitted in that slot.
5. *Retransmission of collisions.* Assume that each packet involved in a collision must be retransmitted in some later slot, with further such retransmissions until the packet is successfully received. A node with a packet that must be retransmitted is said to be *backlogged.*

6a. *No buffering.* If one packet at a node is currently waiting for transmission or colliding with another packet during transmission, new arrivals at that node are discarded and never transmitted. An alternative to this assumption is the following.

6b. *Infinite set of nodes* ($m = \infty$). The system has an infinite set of nodes and each newly arriving packet arrives at a new node.

Discussion of assumptions. The slotted system assumption (1) has two effects. The first is to turn the system into a discrete-time system, thus simplifying analysis. The second is to preclude, for the moment, the possibility of carrier sensing or early collision detection. Carrier sensing is treated in Section 4.4 and early collision detection is treated in Section 4.5. Both allow much more efficient use of the multiaccess channel, but can be understood more clearly as an extension of the present model. Synchronizing the transmitters for slotted arrival at the receiver is not entirely trivial, but may be accomplished with relatively stable clocks, a small amount of feedback from the receiver, and some guard time between the end of a packet transmission and the beginning of the next slot.

The assumption of Poisson arrivals (2) is unrealistic for the case of multipacket messages. We discuss this issue in Section 4.5 in terms of nodes making reservations for use of the channel.

The assumption of collision or perfect reception (3) ignores the possibility of errors due to noise and also ignores the possibility of "capture" techniques, by which a receiver can sometimes capture one transmission in the presence of multiple transmissions.

The assumption of immediate feedback (4) is quite unrealistic, particularly in the case of satellite channels. It is made to simplify the analysis, and we shall see later that delayed feedback complicates multiaccess algorithms but causes no fundamental problems. We also discuss the effects of more limited types of feedback later.

The assumption that colliding packets must be retransmitted (5) is certainly reasonable in providing reliable communication. In light of this assumption, the no-buffering assumption (6a) appears rather peculiar, since new arrivals to backlogged nodes are thrown away with impunity, but packets once transmitted must be retransmitted until successful. In practice, one generally provides some buffering along with some form of flow control to ensure that not too many packets back up at a node. Our interest in this section, however, is in multiaccess channels with a large number of nodes, a relatively

small arrival rate λ, and small required delay (*i.e.*, the conditions under which TDM does not suffice). Under these conditions, the fraction of backlogged nodes is typically small, and new arrivals at backlogged nodes are almost negligible. Thus, the delay for a system without buffering should be relatively close to that with buffering. Also, the delay for the unbuffered system provides a lower bound to the delay for a wide variety of systems with buffering and flow control.

The infinite-node assumption (6b) alternatively provides us with an upper bound to the delay that can be achieved with a finite number of nodes. In particular, given any multiaccess algorithm (*i.e.*, any rule that each node employs for selecting packet transmission times), each of a finite set of nodes can regard itself as a set of virtual nodes, one for each arriving packet. With the application of the given algorithm independently for each such packet, the situation is equivalent to that with an infinite set of nodes (*i.e.*, assumption 6b). In this approach, a node with several backlogged packets will sometimes send multiple packets in one slot, causing a sure collision. Thus, it appears that by avoiding such sure collisions and knowing the number of buffered packets at a node, a system with a finite number of nodes and buffering could achieve smaller delays than with $m = \infty$; in any case, however, the $m = \infty$ delay can always be achieved.

If the performance using assumption 6a is similar to that using 6b, then we are assured that we have a good approximation to the performance of a system with arbitrary assumptions about buffering. From a theoretical standpoint, assumption 6b captures the essence of multiaccess communication much better than 6a. We use 6a initially, however, because it is less abstract and it provides some important insights about the relationship between TDM and other algorithms.

4.2.2 Slotted Aloha

The Aloha network [Abr70] was developed around 1970 to provide radio communication between the central computer and various data terminals at the campuses of the University of Hawaii. This multiaccess approach will be described in Section 4.2.4, but first we discuss an improvement called slotted Aloha [Rob72]. The basic idea of this algorithm is that each unbacklogged node simply transmits a newly arriving packet in the first slot after the packet arrival, thus risking occasional collisions but achieving very small delay if collisions are rare. This approach should be contrasted with TDM, in which, with m nodes, an arriving packet would have to wait for an average of $m/2$ slots for its turn to transmit. Thus, slotted Aloha transmits packets almost immediately with occasional collisions, whereas TDM avoids collisions at the expense of large delays.

When a collision occurs in slotted Aloha, each node sending one of the colliding packets discovers the collision at the end of the slot and becomes backlogged. If each backlogged node were simply to retransmit in the next slot after being involved in a collision, then another collision would surely occur. Instead, such nodes wait for some random number of slots before retransmitting.

To gain some initial intuitive understanding of slotted Aloha, we start with an instructive but flawed analysis. With the infinite-node assumption (*i.e.*, 6b), the number

of new arrivals transmitted in a slot is a Poisson random variable with parameter λ. If the retransmissions from backlogged nodes are sufficiently randomized, it is plausible to approximate the total number of retransmissions and new transmissions in a given slot as a Poisson random variable with some parameter $G > \lambda$. With this approximation, the probability of a successful transmission in a slot is Ge^{-G}. Finally, in equilibrium, the arrival rate, λ, to the system should be the same as the departure rate, Ge^{-G}. This relationship is illustrated in Fig. 4.2.

We see that the maximum possible departure rate (according to the argument above) occurs at $G = 1$ and is $1/e \approx 0.368$. We also note, somewhat suspiciously, that for any arrival rate less than $1/e$, there are two values of G for which the arrival rate equals the departure rate. The problem with this elementary approach is that it provides no insight into the dynamics of the system. As the number of backlogged packets changes, the parameter G will change; this leads to a feedback effect, generating further changes in the number of backlogged packets. To understand these dynamics, we will have to analyze the system somewhat more carefully. The simple picture below, however, correctly identifies the maximum throughput rate of slotted Aloha as $1/e$ and also shows that G, the mean number of attempted transmissions per slot, should be on the order of 1 to achieve a throughput close to $1/e$. If $G < 1$, too many idle slots are generated, and if $G > 1$, too many collisions are generated.

To construct a more precise model, assume that each backlogged node retransmits with some fixed probability q_r in each successive slot until a successful transmission occurs. In other words, the number of slots from a collision until a given node involved in the collision retransmits is a geometric random variable having value $i \geq 1$ with probability $q_r(1 - q_r)^{i-1}$. The original version of slotted Aloha employed a uniform distribution for retransmission, but this is more difficult to analyze and has no identifiable

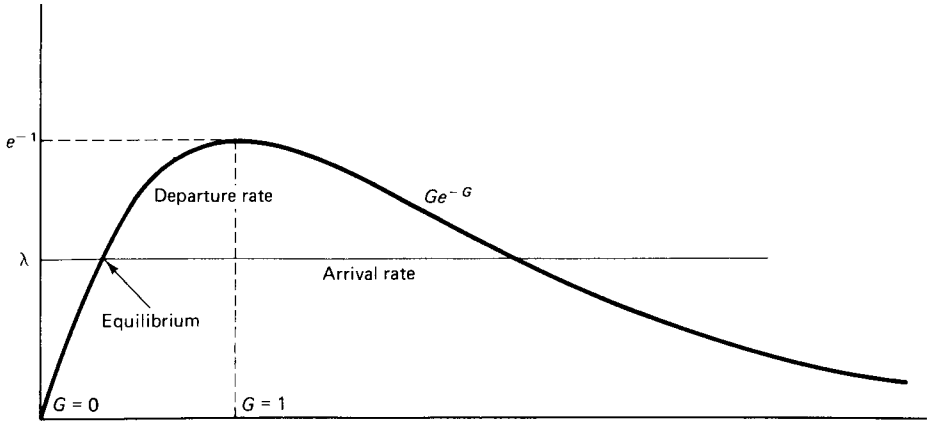


Figure 4.2 Departure rate as a function of attempted transmission rate G for slotted Aloha. Ignoring the dynamic behavior of G, departures (successful transmissions) occur at a rate Ge^{-G}, and arrivals occur at a rate λ, leading to a hypothesized equilibrium point as shown.

advantages over the geometric distribution. We will use the no-buffering assumption (6a) and switch later to the infinite node assumption (6b).

The behavior of slotted Aloha can now be described as a discrete-time Markov chain. Let n be the number of backlogged nodes at the beginning of a given slot. Each of these nodes will transmit a packet in the given slot, independently of each other, with probability q_r. Each of the $m-n$ other nodes will transmit a packet in the given slot if one (or more) such packets arrived during the previous slot. Since such arrivals are Poisson distributed with mean λ/m, the probability of no arrivals is $e^{-\lambda/m}$; thus, the probability that an unbacklogged node transmits a packet in the given slot is $q_a = 1 - e^{-\lambda/m}$. Let $Q_a(i, n)$ be the probability that i unbacklogged nodes transmit packets in a given slot, and let $Q_r(i, n)$ be the probability that i backlogged nodes transmit,

$$Q_a(i, n) = \binom{m-n}{i} (1 - q_a)^{m-n-i} q_a^i \tag{4.1}$$

$$Q_r(i, n) = \binom{n}{i} (1 - q_r)^{n-i} q_r^i \tag{4.2}$$

Note that from one slot to the next, the state (*i.e.*, the number of backlogged packets) increases by the number of new arrivals transmitted by unbacklogged nodes, less one if a packet is transmitted successfully. A packet is transmitted successfully, however, only if one new arrival and no backlogged packet, or no new arrival and one backlogged packet is transmitted. Thus, the state transition probability of going from state n to $n + i$ is given by

$$P_{n,n+i} = \begin{cases} Q_a(i, n), & 2 \le i \le (m-n) \\ Q_a(1, n)[1 - Q_r(0, n)], & i = 1 \\ Q_a(1, n)Q_r(0, n) - Q_a(0, n)[1 - Q_r(1, n)], & i = 0 \\ Q_a(0, n)Q_r(1, n), & i = -1 \end{cases} \tag{4.3}$$

Figure 4.3 illustrates this Markov chain. Note that the state can decrease by at most 1 in a single transition, and this makes it easy to calculate the steady-state probabilities

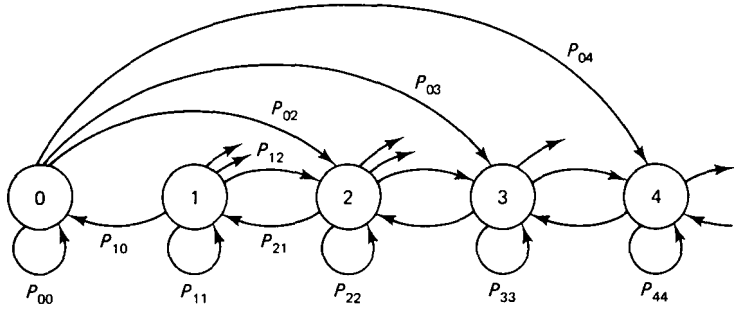


Figure 4.3 Markov chain for slotted Aloha. The state (*i.e.*, backlog) can decrease by at most one per transition, but can increase by an arbitrary amount.

iteratively, finding p_n for each successively larger n in terms of p_0 and finally, finding p_0 as a normalizing constant (see Problem 4.1). From this, the expected number of backlogged nodes can be found, and from Little's theorem, the average delay can be calculated.

Unfortunately, this system has some very strange properties for a large number of nodes, and the steady-state results above are very misleading. To get some intuitive understanding of this, note that we would like to choose the retransmission probability q_r to be moderately large, so as to avoid large delays after collisions. If the arrival rate is small and not many packets are involved in collisions, this works well and retransmissions are normally successful. On the other hand, if the system is afflicted with a run of bad luck and the number of backlogged packets n gets large enough to satisfy $q_r n >> 1$, then collisions occur in almost all successive slots and the system remains heavily backlogged for a long time.

To understand this phenomenon quantitatively, define the *drift* in state n (D_n) as the expected change in backlog over one slot time, starting in state n. Thus, D_n is the expected number of new arrivals accepted into the system [*i.e.*, $(m-n)q_a$] less the expected number of successful transmissions in the slot; the expected number of successful transmissions is just the probability of a successful transmission, defined as P_{succ}. Thus,

$$D_n = (m-n)q_a - P_{succ} \tag{4.4}$$

where

$$P_{succ} = Q_a(1,n)Q_r(0,n) + Q_a(0,n)Q_r(1,n) \tag{4.5}$$

Define the attempt rate $G(n)$ as the expected number of attempted transmissions in a slot when the system is in state n, that is,

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

If q_a and q_r are small, P_{succ} is closely approximated as the following function of the attempt rate:

$$P_{succ} \approx G(n)e^{-G(n)} \tag{4.6}$$

This approximation can be derived directly from Eq. (4.5), using the approximation $(1-x)^y \approx e^{-xy}$ for small x in the expressions for Q_a and Q_r. Similarly, the probability of an idle slot is approximately $e^{-G(n)}$. Thus, the number of packets in a slot is well approximated as a Poisson random variable (as in the earlier intuitive analysis), but the parameter $G(n)$ varies with the state. Figure 4.4 illustrates Eqs. (4.4) and (4.6) for the case $q_r > q_a$ (this is the only case of interest, as discussed later). The drift is the difference between the curve and the straight line. Since the drift is the expected change in state from one slot to the next, the system, although perhaps fluctuating, tends to move in the direction of the drift and consequently tends to cluster around the two stable points with rare excursions between the two.

There are two important conclusions from this figure. First, the departure rate (*i.e.*, P_{succ}) is at most $1/e$ for large m. Second, the departure rate is almost zero for long periods whenever the system jumps to the undesirable stable point. Consider the effect of

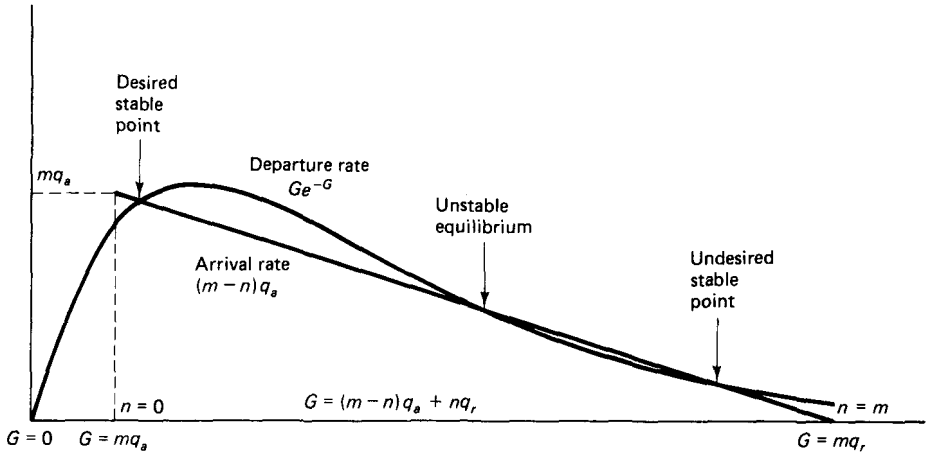


Figure 4.4 Instability of slotted Aloha. The horizontal axis corresponds to both the state n and the attempt rate G, which are related by the linear equation $G = (m-n)q_a + nq_r$ with $q_r > q_a$. For n to the left of the unstable point, D is negative and n drifts toward the desired stable point. For n to the right of the unstable point, D is positive and n drifts toward the undesired stable point.

changing q_r. As q_r is increased, the delay in retransmitting a collided packet decreases, but also the linear relationship between n and the attempt rate $G(n) = (m-n)q_a + nq_r$ changes [*i.e.*, $G(n)$ increases with n faster when q_r is increased]. If the horizontal scale for n is held fixed in Fig. 4.4, this change in attempt rate corresponds to a contraction of the horizontal $G(n)$ scale, and thus to a horizontal contraction of the curve Ge^{-G}. This means that the number of backlogged packets required to exceed the unstable equilibrium point decreases. Alternatively, if q_r is decreased, retransmission delay increases, but it becomes more difficult to exceed the unstable equilibrium point. If q_r is decreased enough (while still larger than q_a), the curve Ge^{-G} will expand enough in Fig. 4.4 that only one stable state will remain. At this stable point, and similarly when $q_r = q_a$, the backlog is an appreciable fraction of m, and this means both that an appreciable number of arriving packets are discarded and that the delay is excessively large.

The question of what values of q_r and arrival rate lead to stable behavior, particularly when q_r and arrival rate vary from node to node and infinite buffering is provided at each node, has received considerable theoretical attention (*e.g.*, [Tsy85]). These theoretical studies generally assume that nodes are considered backlogged immediately on arrival. Problem 4.8, however, illustrates that if q_r is small enough for stable operation, then the delay is considerably greater than that with TDM; thus these approaches are not of great practical importance.

If we replace the no-buffering assumption with the infinite-node assumption, the attempt rate $G(n)$ becomes $\lambda + nq_r$ and the straight line representing arrivals in Fig. 4.4 becomes horizontal. In this case, the undesirable stable point disappears, and once the state of the system passes the unstable equilibrium, it tends to increase without bound. In this case, the corresponding infinite-state Markov chain has no steady-state distribution

and the expected backlog increases without bound as the system continues to run. (See Section 3A.5 for further discussion of stability of such systems.)

From a practical standpoint, if the arrival rate λ is very much smaller than $1/e$, and if q_r is moderate, then the system could be expected to remain in the desired stable state for very long periods. In the unlikely event of a move to the undesired stable point, the system could be restarted with backlogged packets lost. Rather than continuing to analyze this rather flawed system, however, we look at modifications of slotted Aloha that cure this stability issue.

4.2.3 Stabilized Slotted Aloha

One simple approach to achieving stability should be almost obvious now. P_{succ} is approximately $G(n)e^{-G(n)}$, which is maximized at $G(n) = 1$. Thus, it is desirable to change q_r dynamically to maintain the attempt rate $G(n)$ at 1. The difficulty here is that n is unknown to the nodes and can only be estimated from the feedback. There are many strategies for estimating n or the appropriate value of q_r (*e.g.*, [HaL82] and [Riv85]). All of them, in essence, increase q_r when an idle slot occurs and decrease q_r when a collision occurs.

Stability and maximum throughput. The notion of stability must be clarified somewhat before proceeding. Slotted Aloha was called unstable in the last subsection on the basis of the representation in Fig. 4.4. Given the no-buffering assumption, however, the system has a well-defined steady-state behavior for all arrival rates. With the infinite-node assumption, on the other hand, there is no steady-state distribution and the expected delay grows without bound as the system continues to run. With the no-buffering assumption, the system discards large numbers of arriving packets and has a very large but finite delay, whereas with the infinite-node assumption, no arrivals are discarded but the delay becomes infinite.

In the following, we shall use the infinite-node assumption (6b), and define a multiaccess system as stable for a given arrival rate if the expected delay per packet (either as a time average or an ensemble average in the limit of increasing time) is finite. Ordinary slotted Aloha is unstable, in this sense, for any arrival rate greater than zero. Note that if a system is stable, then for a sufficiently large but finite number of nodes m, the system (regarding each arrival as corresponding to a new virtual node) has a smaller expected delay than TDM, since the delay of TDM, for a fixed overall arrival rate, increases linearly with m.

The maximum stable throughput of a multiaccess system is defined as the least upper bound of arrival rates for which the system is stable. For example, the maximum stable throughput of ordinary slotted Aloha is zero. Our purpose with these definitions is to study multiaccess algorithms that do not require knowledge of the number of nodes m and that maintain small delay (for given λ) independent of m. Some discussion will be given later to modifications that make explicit use of the number of nodes.

Returning to the stabilization of slotted Aloha, note that when the estimate of backlog is perfect, and $G(n)$ is maintained at the optimal value of 1, then (according to

the Poisson approximation) idles occur with probability $1/e \approx 0.368$, successes occur with probability $1/e$, and collisions occur with probability $1 - 2/e \approx 0.264$. Thus, the rule for changing q_r should allow fewer collisions than idles. The maximum stable throughput of such a system is at most $1/e$. To see this, note that when the backlog is large, the Poisson approximation becomes more accurate, the success rate is then limited to $1/e$, and thus the drift is positive for $\lambda > 1/e$. It is important to observe that this argument depends on all backlogged nodes using the same retransmission probability. We shall see in Section 4.3 that if nodes use both their own history of retransmissions and the feedback history in their decisions about transmitting, maximum stable throughputs considerably higher than $1/e$ are possible.

Pseudo-Bayesian algorithm. Rivest's pseudo-Bayesian algorithm [Riv85] is a particularly simple and effective way to stabilize Aloha. This algorithm is essentially the same as an earlier, independently derived algorithm by Mikhailov [Mik79], but Rivest's Bayesian interpretation simplifies understanding. The algorithm differs from slotted Aloha in that new arrivals are regarded as backlogged immediately on arrival. Rather than being transmitted with certainty in the next slot, they are transmitted with probability q_r in the same way as packets involved in previous collisions. Thus, if there are n backlogged packets (including new arrivals) at the beginning of a slot, the attempt rate is $G(n) = nq_r$, and the probability of a successful transmission is $nq_r(1 - q_r)^{n-1}$. For unstabilized Aloha, this modification would not make much sense, since q_r has to be relatively small and new arrivals would be unnecessarily delayed. For stabilized Aloha, however, q_r can be as large as 1 when the estimated backlog is negligible, so that new arrivals are held up only when the system is already estimated to be congested. In Problem 4.6 it is shown that this modification increases the probability of success if the backlog estimate is accurate.

The algorithm operates by maintaining an estimate $\hat{n}$ of the backlog n at the beginning of each slot. Each backlogged packet is then transmitted (independently) with probability $q_r(\hat{n}) = \min\{1, 1/\hat{n}\}$. The minimum operation limits q_r to at most 1, and subject to this limitation, tries to achieve an attempt rate $G = nq_r$ of 1. For each k, the estimated backlog at the beginning of slot $k + 1$ is updated from the estimated backlog and feedback for slot k according to the rule

$$\hat{n}_{k+1} = \begin{cases} \max\{\lambda, \hat{n}_k + \lambda - 1\}, & \text{for idle or success} \\ \hat{n}_k + \lambda + (e-2)^{-1}, & \text{for collision} \end{cases} \tag{4.7}$$

The addition of λ to the previous backlog accounts for new arrivals, and the max operation ensures that the estimate is never less than the contribution from new arrivals. On successful transmissions, 1 is subtracted from the previous backlog to account for the successful departure. Finally, subtracting 1 from the previous backlog on idle slots and adding $(e-2)^{-1}$ on collisions has the effect of decreasing $\hat{n}$ when too many idles occur and increasing $\hat{n}$ when too many collisions occur. For large backlogs, if $\hat{n} = n$, each of the n backlogged packets is independently transmitted in a slot with probability $q_r = 1/n$. Thus $G(n)$ is 1, and, by the Poisson approximation, idles occur with probability $1/e$ and

collisions with probability $(e-2)/e$, so that decreasing $\hat{n}$ by 1 on idles and increasing $\hat{n}$ by $(e-2)^{-1}$ on collisions maintains the balance between n and $\hat{n}$ on the average.

It is shown in Problem 4.9 that if the a priori probability distribution on n_k is Poisson with parameter $\hat{n}_k \geq 1$, then given an idle or successful transmission in slot k, the probability distribution of n_{k+1} is Poisson with parameter $\hat{n}_k + \lambda - 1$. Given a collision, the resulting distribution of n_{k+1} is not quite Poisson but is reasonably approximated as Poisson with parameter $\hat{n}_{k+1}$. For this reason, the algorithm is called pseudo-Bayesian.

Figure 4.5 provides some heuristic understanding of why the algorithm is stable for all $\lambda < 1/e$. The state of the system is characterized by n and $\hat{n}$, and the figure shows the expected drift in these variables from one slot to the next. If n and $\hat{n}$ are large and equal, the expected drift of each is $\lambda - 1/e$, which is negative. On the other hand, if the absolute value of $n - \hat{n}$ is large, the expected drift of n is positive, but the expected reduction in $|n - \hat{n}|$ is considerably larger. Thus, if the system starts at some arbitrary point in the $(n, \hat{n})$ plane, n might increase on the average for a number of slots, but eventually n and $\hat{n}$ will come closer together and then n will decrease on the average.

In applications, the arrival rate λ is typically unknown and slowly varying. Thus, the algorithm must either estimate λ from the time-average rate of successful transmissions or set its value within the algorithm to some fixed value. It has been shown by Mikhailov (see [Kel85]) and Tsitsiklis [Tsi87] that if the fixed value $1/e$ is used within the algorithm, stability is achieved for all actual $\lambda < 1/e$. Nothing has been proven about the behavior of the algorithm when a dynamic estimate of λ is used within the algorithm.

Approximate delay analysis. An exact analysis of expected delay for this algorithm, even with known λ, appears to be very difficult, but it is instructive to analyze an approximate model. Assume that λ is known and that the probability of successful transmission P_{succ} is $1/e$ whenever the backlog n is 2 or more, and that $P_{succ} = 1$ for

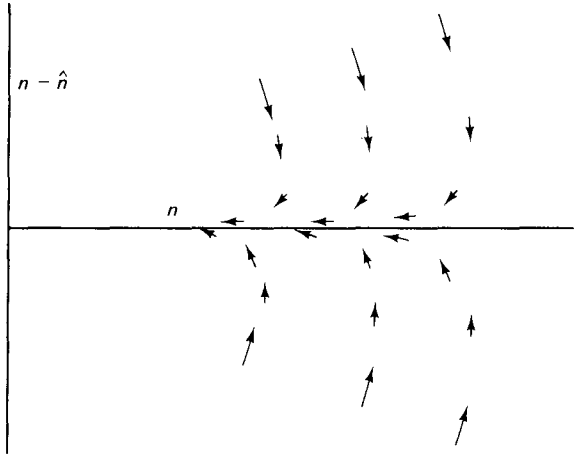


Figure 4.5 Drift of n and $n - \hat{n}$ for the pseudo-Bayesian stabilization algorithm. When the absolute value of $n - \hat{n}$ is large, it approaches 0 faster than n increases.

$n = 1$. This is a reasonable model for very small λ, since very few collisions occur and q_r is typically 1. It is also reasonable for large $\lambda < 1/e$, since typically n is large and $\hat{n} \approx n$.

Let W_i be the delay from the arrival of the i^{th} packet until the beginning of the i^{th} successful transmission. Note that if the system were first-come first-serve, W_i would be the queueing time of the i^{th} arrival. By the same argument as in Problem 3.32, however, the average of W_i over all i is equal to the expected queueing delay W. Let n_i be the number of backlogged packets at the instant before i's arrival; n_i does not include any packet currently being successfully transmitted, but does include current unsuccessful transmissions. W_i can be expressed as

$$W_i = R_i + \sum_{j=1}^{n_i} t_j + y_i \tag{4.8}$$

R_i is the residual time to the beginning of the next slot, and t_1 (for $n_i > 0$) is the subsequent interval until the next successful transmission is completed. Similarly, t_j, $1 < j \leq n_i$, is the interval from the end of the $(j-1)^{\text{th}}$ subsequent success to the end of the j^{th} subsequent success. After those n_i successes, y_i is the remaining interval until the beginning of the next successful transmission (*i.e.*, the i^{th} transmission overall).

Observe that for each interval t_j, the backlog is at least two, counting the new i^{th} arrival and the n_i packets already backlogged. Thus, each slot is successful with probability $1/e$, and the expected value of each t_j is e. Next, observe that Little's formula relates the expected values of W_i and n_i (*i.e.*, W_i is the wait not counting the successful transmission, and n_i is the number in the system not counting successful transmissions in process). Finally, the expected value of R_i is $1/2$ (counting time in slots). Thus, taking the expected value of Eq. (4.8) and averaging over i, we get

$$W = 1/2 + \lambda e W + E\{y\} \tag{4.9}$$

Now consider the system at the first slot boundary at which both the $(i-1)^{\text{st}}$ departure has occurred and the i^{th} arrival has occurred. If the backlog is 1 at that point (*i.e.*, only the i^{th} packet is in the system), then y_i is 0. Alternatively, if $n > 1$, then $E\{y_i\} = e - 1$. Let p_n be the steady-state probability that the backlog is n at a slot boundary. Then, since a packet is always successfully transmitted if the state is 1 at the beginning of the slot, we see that p_1 is the fraction of slots in which the state is 1 and a packet is successfully transmitted. Since λ is the total fraction of slots with successful transmissions, p_1/λ is the fraction of packets transmitted from state 1 and $1 - p_1/\lambda$ is the fraction transmitted from higher-numbered states. It follows that

$$E\{y\} = \frac{(e-1)(\lambda - p_1)}{\lambda} \tag{4.10}$$

Finally, we must determine p_1. From the argument above, we see that $\lambda = p_1 + (1 - p_0 - p_1)/e$. Also, state 0 can be entered at a slot boundary only if no arrivals occurred in the previous slot and the previous state was 0 or 1. Thus, $p_0 = (p_0 + p_1)e^{-\lambda}$. Solving for p_1 gives

$$p_1 = \frac{(1 - \lambda e)(e^{\lambda} - 1)}{1 - (e-1)(e^{\lambda} - 1)} \tag{4.11}$$

Combining Eqs. (4.9) to (4.11) yields

$$W = \frac{e - 1/2}{1 - \lambda e} - \frac{(e^{\lambda} - 1)(e - 1)}{\lambda[1 - (e - 1)(e^{\lambda} - 1)]} \tag{4.12}$$

This value of W is quite close to delay values for the pseudo-Bayesian algorithm obtained through simulation. Figure 4.6 plots $E\{W\}$ for this approximate model and compares it with the queueing delay for TDM with 8 and 16 nodes. It is quite surprising that the delay is so small even for arrival rates relatively close to $1/e$. It appears that the assumption of immediate feedback is unnecessary for stabilization strategies of this type to be stable; the argument is that feedback delay will make the estimate $\hat{n}$ less accurate, but $n/\hat{n}$ will still stay close to 1 for large n.

Binary exponential backoff. In the packet radio networks to be discussed in Section 4.6, and in some other multiaccess situations, the assumption of 0,1,e feedback on all slots is unrealistic. In some systems, a node receives feedback only about whether or not its own packets were successfully transmitted; it receives no feedback about slots in which it does not transmit. Such limited feedback is sufficient for slotted Aloha but is insufficient for the backlog estimation of the pseudo-Bayesian strategy. Binary exponential backoff [MeB76] is a stabilization strategy used in Ethernet that employs only this more limited form of feedback.

This strategy is very simple; if a packet has been transmitted unsuccessfully i times, then the probability of transmission in successive slots is set at $q_r = 2^{-i}$ (or is uniformly distributed over the next 2^i slots after the i^{th} failure). When a packet initially arrives in the system, it is transmitted immediately in the next slot.

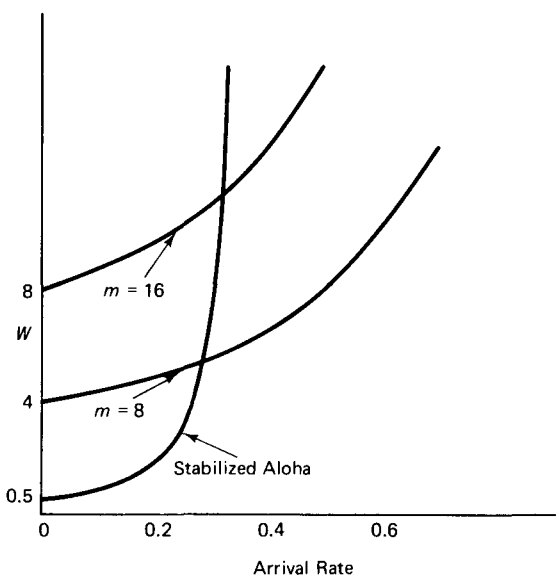


Figure 4.6 Comparison of expected waiting time W in slots, from arrival until beginning of successful transmission, for stabilized Aloha and for TDM with $m = 8$ and $m = 16$. For small arrival rates, the delay of stabilized Aloha is little more than waiting for the next slot, whereas as the arrival rate approaches $1/e$, the delay becomes unbounded.

Some rationale for this strategy can be seen by noting that when a packet first arrives (with this limited feedback), the node knows nothing of the backlog, so the immediate first transmission is reasonable. With successive collisions, any reasonable estimate of backlog would increase, motivating the decrease in the local q_r. To make matters worse, however, as q_r is reduced, the node gets less feedback per slot about the backlog, and thus, to play safe, it is reasonable to increase the backlog estimate by larger and larger amounts on each successive collision.

Unfortunately, in the limit as the number of nodes approaches infinity, this strategy is unstable for every arrival rate λ greater than 0 [Ald86]. It is unknown whether or not any strategy can achieve stability with this type of limited feedback. Problem 4.10, however, develops another strategy that can be used with this limited feedback and with a finite number of nodes with unlimited buffering to achieve finite expected delay for any λ less than 1. Unfortunately, the price of this high throughput is inordinately large delay.

4.2.4 Unslotted Aloha

Unslotted, or pure, Aloha [Abr70] was the precursor to slotted Aloha. In this strategy, each node, upon receiving a new packet, transmits it immediately rather than waiting for a slot boundary. Slots play no role in pure Aloha, so we temporarily omit the slotted system assumption. If a packet is involved in a collision, it is retransmitted after a random delay. Assume that if the transmission times for two packets overlap at all, the CRCs on those packets will fail and retransmission will be required. We assume that the receiver rebroadcasts the composite received signal (or that all nodes receive the composite signal), so that each node, after a given propagation delay, can determine whether or not its transmitted packets were correctly received. Thus, we have the same type of limited feedback discussed in the last subsection. Other types of feedback could be considered and Problem 4.11 develops some of the peculiar effects of other feedback assumptions.

Figure 4.7 shows that if one packet starts transmission at time t, and all packets have unit length, then any other transmission starting between $t-1$ and $t+1$ will cause a collision. For simplicity, assume an infinite number of nodes (*i.e.*, assumption 6b) and let n be the number of backlogged nodes at a given time. For our present purposes, a node is considered backlogged from the time it has determined that its previously transmitted packet was involved in a collision until the time that it attempts retransmission. Assume that the period until attempted retransmission τ is an exponentially distributed random variable with probability density $xe^{-x\tau}$, where x is an arbitrary parameter interpreted as a node's retransmission attempt rate. Thus, with an overall Poisson arrival rate of λ to the system, the initiation times of attempted transmissions is a time-varying Poisson process of rate $G(n) = \lambda + nx$ in which n is the backlog at a given time.

Consider the sequence of successive transmission attempts on the channel. For some given i, let τ_i be the duration of the interval between the initiations of the i^{th} and $(i+1)^{\text{th}}$ transmission attempt. The i^{th} attempt will be successful if both τ_i and τ_{i-1} exceed 1 (assuming unit length packets). Given the backlog in each intertransmission

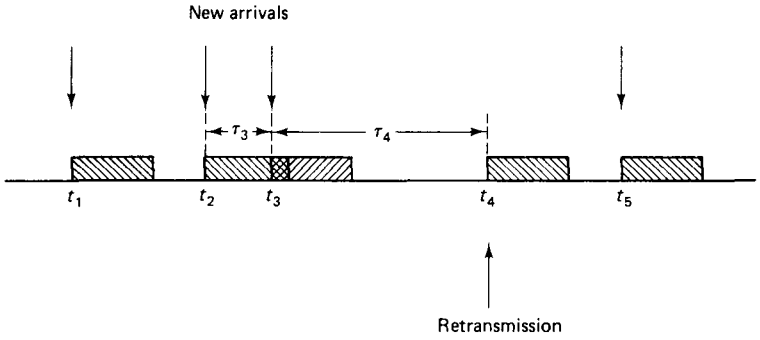


Figure 4.7 Unslotted Aloha. New arrivals are transmitted immediately and unsuccessful transmissions are repeated after a random delay. Packet transmission time is one unit, and two transmissions collide if their interdeparture interval is less than one unit.

interval, these intervals are independent. Thus, assuming a backlog of n for each interval, we have

$$P_{succ} = e^{-2G(n)} \tag{4.13}$$

Since attempted transmissions occur at rate $G(n)$, the throughput (*i.e.*, the expected number of successful transmissions per unit time) as a function of n is

$$\text{throughput}\,(n) = G(n)e^{-2G(n)} \tag{4.14}$$

Figure 4.8 illustrates this result. The situation is very similar to that of slotted Aloha, except that the throughput has a maximum of $1/(2e)$, achieved when $G(n) = 1/2$. The analysis above is approximate in the sense that Eq. (4.13) assumes that the backlog is the same in the intervals surrounding a given transmission attempt, whereas according to our definition of backlog, the backlog decreases by one whenever a backlogged packet initiates transmission and increases by one whenever a collided packet is detected. For small x (*i.e.*, large mean time before attempted retransmission), this effect is relatively small.

It can be seen from Fig. 4.8 that pure Aloha has the same type of stability problem as slotted Aloha. For the limited feedback assumed here, stability is quite difficult to achieve or analyze (as is the case with slotted Aloha). Very little is known about stabilization for pure Aloha, but if λ is very small and the mean retransmission time very large, the system can be expected to run for long periods without major backlog buildup.

One of the major advantages of pure Aloha is that it can be used with variable-length packets, whereas with slotted Aloha, long packets must be broken up to fit into slots and short packets must be padded out to fill up slots. This compensates for some of the inherent throughput loss of pure Aloha and gives it an advantage in simplicity. Some analysis, with simplifying assumptions, of the effect of variable-length packets appears in [Fer75] and [San80].

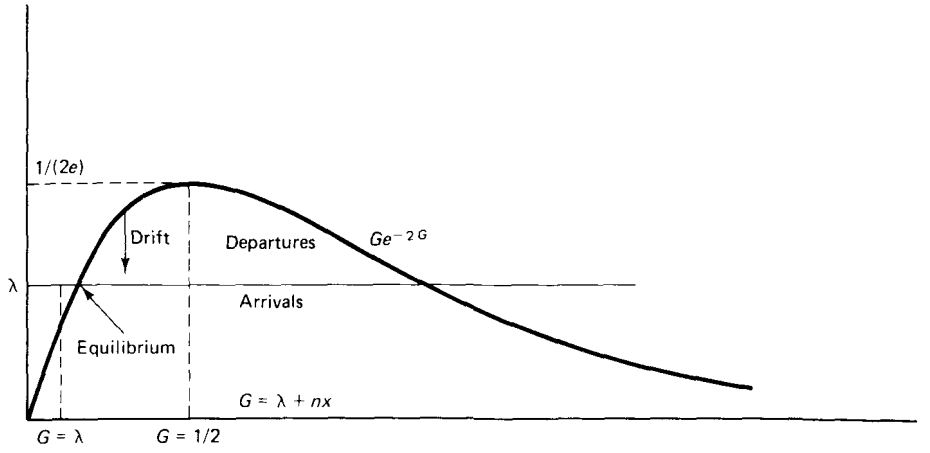


Figure 4.8 Pure Aloha as a function of the attempted transmission rate G. Successful departures leave the system at a rate Ge^{-2G}, and arrivals occur at a rate λ, leading to a hypothesized equilibrium at the point shown.

4.3 SPLITTING ALGORITHMS

We have seen that slotted Aloha requires some care for stabilization and is also essentially limited to throughputs of $1/e$. We now want to look at more sophisticated collision resolution techniques that both maintain stability without any complex estimation procedures and also increase the achievable throughput. To get an intuitive idea of how this can be done, note that with relatively small attempt rates, when a collision occurs, it is most likely between only two packets. Thus, if new arrivals could be inhibited from transmission until the collision was resolved, each of the colliding packets could be independently retransmitted in the next slot with probability $1/2$. This would lead to a successful transmission for one of the packets with probability $1/2$, and the other could then be transmitted in the following slot. Alternatively, with probability $1/2$, another collision or an idle slot occurs. In this case, each of the two packets would again be independently transmitted in the next slot with probability $1/2$, and so forth until a successful transmission occurred, which would be followed by the transmission of the remaining packet.

With the strategy above, the two packets require two slots with probability $1/2$, three slots with probability $1/4$, and i slots with probability $2^{-(i-1)}$. The expected number of slots for sending these two packets can thus be calculated to be three, yielding a throughput of $2/3$ for the period during which the collision is being resolved.

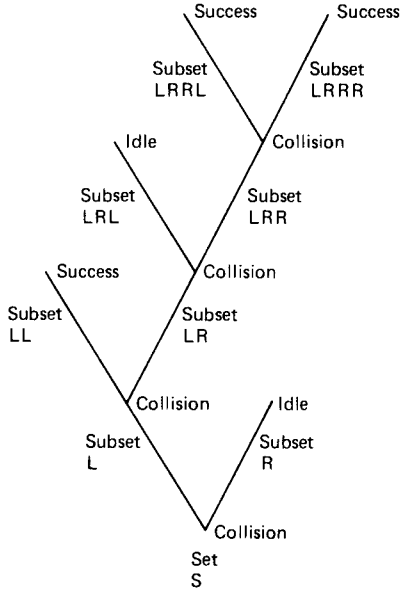
There are various ways in which the nodes involved in a collision could choose whether or not to transmit in successive slots. Each node could simply flip an unbiased coin for each choice. Alternatively (in a way to be described precisely later) each node could use the arrival time of its collided packet. Finally, assuming a finite set of nodes, each with a unique identifier represented by a string of bits, a node could use the successive bits of its identity to make the successive choices. This last alternative has the

advantage of limiting the number of slots required to resolve a collision, since each pair of nodes must differ in at least one bit of their identifiers. All of these alternatives have the common property that the set of colliding nodes is split into subsets, one of which transmits in the next slot. If the collision is not resolved (*e.g.*, if each colliding node is in the same subset), then a further splitting into subsets takes place. We call algorithms of this type *splitting algorithms*. In the subsequent development of these algorithms, we assume a slotted channel, Poisson arrivals, collisions or perfect reception, $(0, 1, e)$ immediate feedback, retransmission of collisions, and an infinite set of nodes (*i.e.*, the assumptions 1 to 6b of Section 4.2.1).

4.3.1 Tree Algorithms

The first splitting algorithms were algorithms with a tree structure ([Cap77], [TsM78], and [Hay76]). When a collision occurs, say in the k^{th} slot, all nodes not involved in the collision go into a waiting mode, and all those involved in the collision split into two subsets (*e.g.*, by each flipping a coin). The first subset transmits in slot $k + 1$, and if that slot is idle or successful, the second subset transmits in slot $k + 2$ (see Fig. 4.9). Alternatively, if another collision occurs in slot $k + 1$, the first of the two subsets splits again, and the second subset waits for the resolution of that collision.

The rooted binary tree structure in Fig. 4.9 represents a particular pattern of idles, successes, and collisions resulting from such a sequence of splittings. S represents the



Slot	Xmit Set	Waiting Sets	Feedback
1	S	–	*e*
2	L	R	*e*
3	LL	LR, R	1
4	LR	R	*e*
5	LRL	LRR, R	0
6	LRR	R	*e*
7	LRRL	LRRR, R	1
8	LRRR	R	1
9	R	–	0

Figure 4.9 Tree algorithm. After a collision, all new arrivals wait and all nodes involved in the collision divide into subsets. Each successive collision in a subset causes that subset to again split into smaller subsets while other nodes wait.

set of packets in the original collision, and L (left) and R (right) represent the two subsets that S splits into. Similarly, LL and LR represent the two subsets that L splits into after L generates a collision. Each vertex in the tree corresponds to a subset (perhaps empty) of backlogged packets. Vertices whose subsets contain two or more packets have two upward branches corresponding to the splitting of the subset into two new subsets; vertices corresponding to subsets with 0 or 1 packet are leaf vertices of the tree.

The set of packets corresponding to the root vertex S is transmitted first, and after the transmission of the subset corresponding to any nonleaf vertex, the subset corresponding to the vertex on the left branch, and all of its descendant subsets, are transmitted before the subsets of the right branch. Given the immediate feedback we have assumed, it should be clear that each node, in principle, can construct this tree as the $0, 1, e$ feedback occurs; each node can keep track of its own subset in that tree, and thus each node can determine when to transmit its own backlogged packet.

The transmission order above corresponds to that of a stack. When a collision occurs, the subset involved is split, and each resulting subset is pushed on the stack (*i.e.*, each stack element is a subset of nodes); then the subset at the head of the stack (*i.e.*, the most recent subset pushed on the stack) is removed from the stack and transmitted. The list, from left to right, of waiting subsets in Fig. 4.9 corresponds to the stack elements starting at the head for the given slot. Note that a node with a backlogged packet can keep track of when to transmit by a counter determining the position of the packet's current subset on the stack. When the packet is involved in a collision, the counter is set to 0 or 1, corresponding to which subset the packet is placed in. When the counter is 0, the packet is transmitted, and if the counter is nonzero, it is incremented by 1 for each collision and decremented by 1 for each success or idle.

One problem with this tree algorithm is what to do with the new packet arrivals that come in while a collision is being resolved. A collision resolution period (CRP) is defined to be completed when a success or idle occurs and there are no remaining elements on the stack (*i.e.*, at the end of slot 9 in Fig. 4.9). At this time, a new CRP starts using the packets that arrived during the previous CRP. In the unlikely event that a great many slots are required in the previous CRP, there will be many new waiting arrivals, and these will collide and continue to collide until the subsets get small enough in this new CRP. The solution to this problem is as follows: At the end of a CRP, the set of nodes with new arrivals is immediately split into j subsets, where j is chosen so that the expected number of packets per subset is slightly greater than 1 (slightly greater because of the temporary high throughput available after a collision). These new subsets are then placed on the stack and the new CRP starts.

The tree algorithm is now essentially completely defined. Each node with a packet involved in the current CRP keeps track of its position in the stack as described above. All the nodes keep track of the number of elements on the stack and the number of slots since the end of the previous CRP. On the completion of that CRP, each node determines the expected number of waiting new arrivals, determines the new number j of subsets, and those nodes with waiting new arrivals randomly choose one of those j subsets and set their counter for the corresponding stack position.

The maximum throughput available with this algorithm, optimized over the choice of j as a function of expected number of waiting packets, is 0.43 packets per slot [Cap77]; we omit any analysis since we next show some simple ways of improving this throughput.

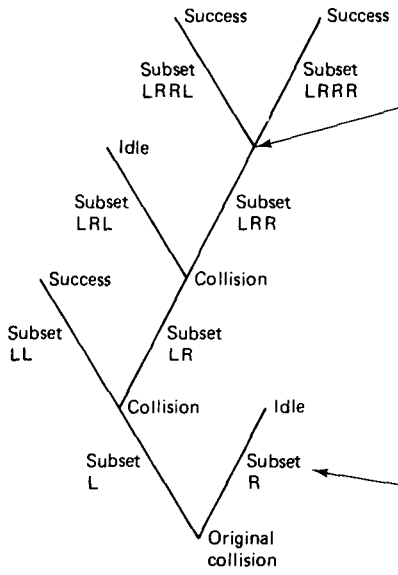
Improvements to the tree algorithm. First consider the situation in Fig. 4.10. Here, in slots 4 and 5, a collision is followed by an idle slot; this means that all the packets involved in the collision were assigned to the second subset. The tree algorithm would simply transmit this second subset, generating a guaranteed collision. An improvement results by omitting the transmission of this second subset, splitting it into two subsets, and transmitting the first subset. Similarly, if an idle again occurs, the second subset is again split before transmission, and so forth.

This improvement can be visualized in terms of a stack and implemented by manipulating counters just like the original tree algorithm. Each node must now keep track of an additional binary state variable that takes the value 1 if, for some $i \geq 1$, the last i slots contained a collision followed by $i - 1$ idles; otherwise, the state variable takes the value 0. If the feedback for the current slot is 0 and the state variable has the value 1, then the state variable maintains the value 1 and the subset on the top of the stack is split into two subsets that are pushed onto the stack in place of the original head element.

The maximum throughput with this improvement is 0.46 packets per slot [Mas80]. In practice, this improvement has a slight problem in that if an idle slot is incorrectly perceived by the receiver as a collision, the algorithm continues splitting indefinitely, never making further successful transmissions. Thus, in practice, after some number h of idle slots followed by splits, the algorithm should be modified simply to transmit the next subset on the stack without first splitting it; if the feedback is very reliable, h can be moderately large, whereas otherwise h should be small.

The next improvement in the tree algorithm not only improves throughput but also greatly simplifies the analysis. Consider what happens when a collision is followed by another collision in the tree algorithm (see slots 1 and 2 of Fig. 4.10). Let x be the number of packets in the first collision, and let x_L and x_R be the number of packets in the resultant subsets; thus, $x = x_L + x_R$. Assume that, a priori, before knowing that there is a collision, x is a Poisson random variable. Visualize splitting these x packets, by coin flip, say, into the two subsets with x_L and x_R packets, respectively, before knowing about the collision. Then a priori x_L and x_R are independent Poisson random variables each with half the mean value of x. Given the two collisions, then, x_L and x_R are independent Poisson random variables conditional on $x_L + x_R \geq 2$ and $x_L \geq 2$. The second condition implies the first, so the first can be omitted; this means that x_R, conditional on the feedback, is still Poisson with its original unconditional distribution. Problem 4.17 demonstrates this result in a more computational and less abstract way. Thus, rather than devoting a slot to this second subset, which has an undesirably small expected number of packets, it is better to regard the second subset as just another part of the waiting new arrivals that have never been involved in a collision.

When the idea above is incorporated into an actual algorithm, the first-come first-serve (FCFS) splitting algorithm, which is the subject of the next subsection, results. Before discussing this, we describe some variants of the tree algorithm.



Slot	Xmit Set	Waiting Sets	Feedback
1	S	–	*e*
2	L	R	*e*
3	LL	LR, R	1
4	LR	R	*e*
5	LRL	LRR, R	0
6	LRRL	LRRR, R	1
7	LRRR	R	1
8	(R)	–	(0)

Figure 4.10 Improvements in the tree algorithm. Subset LRR can be split without first being transmitted since the feedback implies that it contains two or more packets. Also, subset R is better combined with new arrivals since the number of packets in it is Poisson with an undesirably low rate.

Variants of the tree algorithm. The tree algorithm as described above requires all nodes to monitor the channel feedback and to keep track of when each collision resolution period ends. This is a disadvantage if the receivers at nodes are turned off when there are no packets to send. One way to avoid this disadvantage while maintaining the other features of the tree algorithm is to have new arrivals simply join the subset of nodes at the head of the stack. Thus, only currently backlogged nodes need to monitor the channel feedback. Algorithms of this type are called *unblocked stack algorithms*, indicating that new arrivals are not blocked until the end of the current collision resolution period. In contrast, the tree algorithm is often called a blocked stack algorithm.

With the tree algorithm, new arrivals are split into a variable number of subsets at the beginning of each collision resolution period, and then subsets are split into two subsets after each collision. With an unblocked stack algorithm, on the other hand, new arrivals are constantly being added to the subset at the head of the stack, and thus, collisions involve a somewhat larger number of packets on the average. Because of the relatively large likelihood of three or more packets in a collision, higher maximum throughputs can be achieved by splitting collision sets into three subsets rather than two. The maximum throughput thus available for unblocked stack algorithms is 0.40 [MaF85].

4.3.2 First-Come First-Serve Splitting Algorithms

In the second improvement to the tree algorithm described above, the second subset involved in a collision is treated as if it had never been transmitted if the first subset

contains two or more packets. Recall also that a set of colliding packets can be split into two subsets in a variety of ways (*e.g.*, by coin flipping, by node identities, or by arrival time). For our present purposes, the simplest choice is splitting by arrival time. By using this approach, each subset will consist of all packets that arrived in some given interval, and when a collision occurs, that interval will be split into two smaller intervals. By always transmitting the earlier arriving interval first, the algorithm will always transmit successful packets in the order of their arrival, leading to the name first-come first-serve (FCFS).

At each integer time k, the algorithm specifies the packets to be transmitted in slot k (*i.e.*, from k to $k+1$) to be the set of all packets that arrived in some earlier interval, say from $T(k)$ to $T(k)+\alpha(k)$. This interval is called the *allocation interval* for slot k (see Fig. 4.11). We can visualize the packets arriving after $T(k)+\alpha(k)$ as being in a

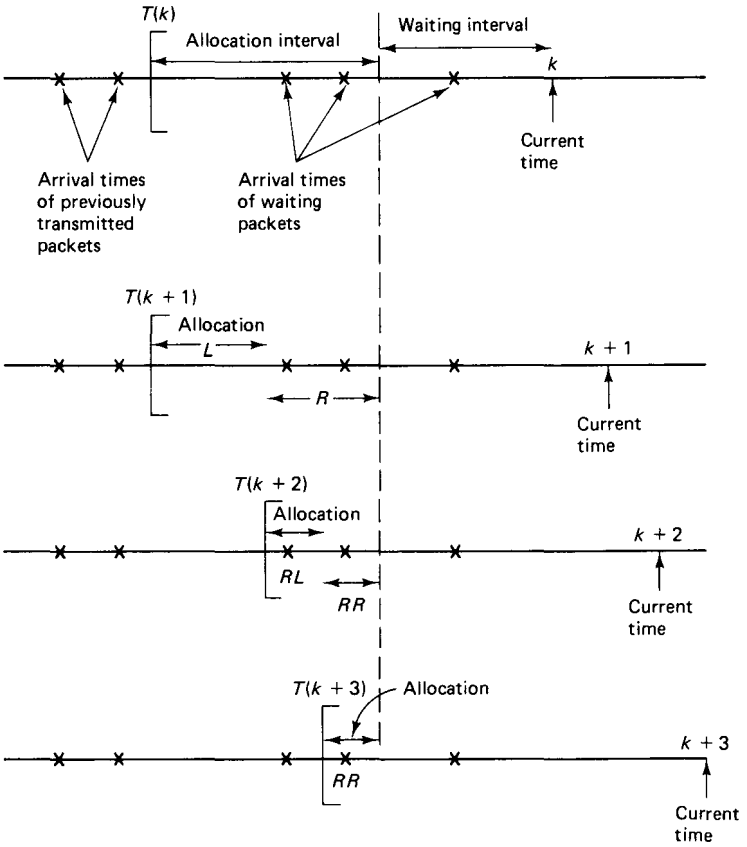


Figure 4.11 FCFS splitting algorithm. Packets are transmitted in order of arrival. On collisions, the allocation interval generating a collision is split into two subintervals, with the leftmost (earlier arrivals) transmitting first.

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 $k-1$.

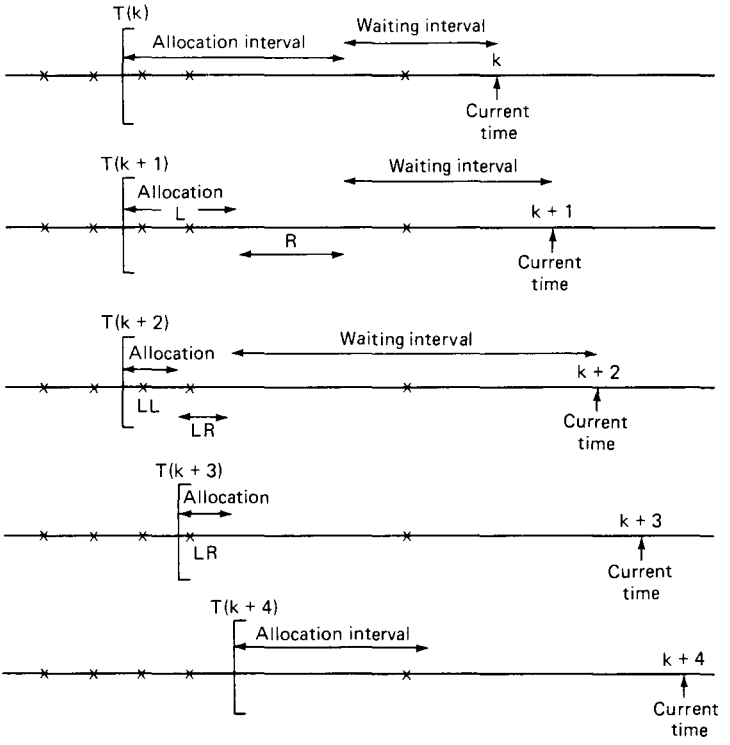


Figure 4.12 FCFS splitting algorithm. When a collision follows another collision, the interval on the right side of the second collision is returned to the waiting interval. The CRP is completed in slot $k + 3$, and a new CRP is started with an allocation interval of fixed size.

If feedback $= e$, then

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

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

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

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

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

$$\alpha(k) = \frac{\alpha(k-1)}{2} \qquad (4.17)$$
$$\sigma(k) = L$$

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

$$T(k) = T(k-1) + \alpha(k-1)$$
$$\alpha(k) = \min[\alpha_0, k - T(k)] \qquad (4.18)$$
$$\sigma(k) = R$$

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

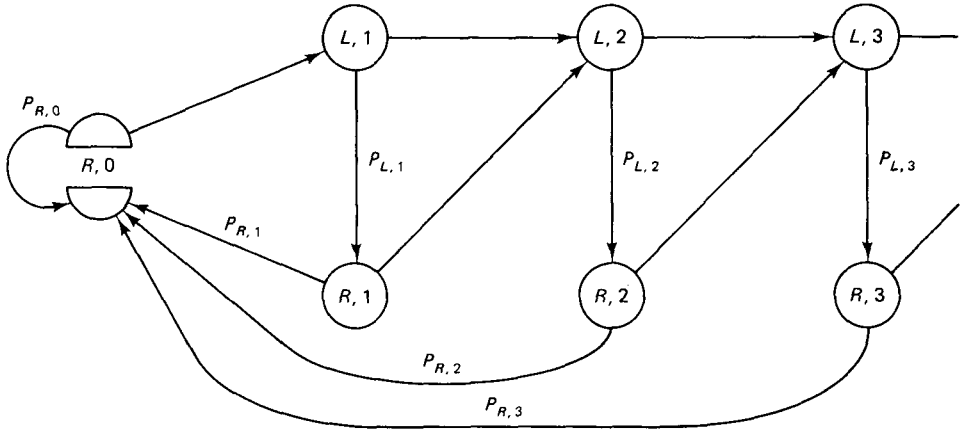


Figure 4.13 Markov chain for FCFS splitting algorithm. The top states are entered after splitting an interval and correspond to the transmission of the left side of that inteval. The lower states are entered after success on the left side and correspond to transmission of the right side. Transitions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

$$G_i = 2^{-i}\lambda\alpha_0 \tag{4.19}$$

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

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

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

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

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

$$P_{R.1} = \frac{P\{x_R = 1\}}{P\{x_R \geq 1\}} = \frac{G_1 e^{-G_1}}{1 - e^{-G_1}} \tag{4.22}$$

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,

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

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

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 $i = 3, 4, \ldots$ (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 $(R, 0)$:

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

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

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

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 $(R, 0)$,

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

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,

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

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

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

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

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

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

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

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.

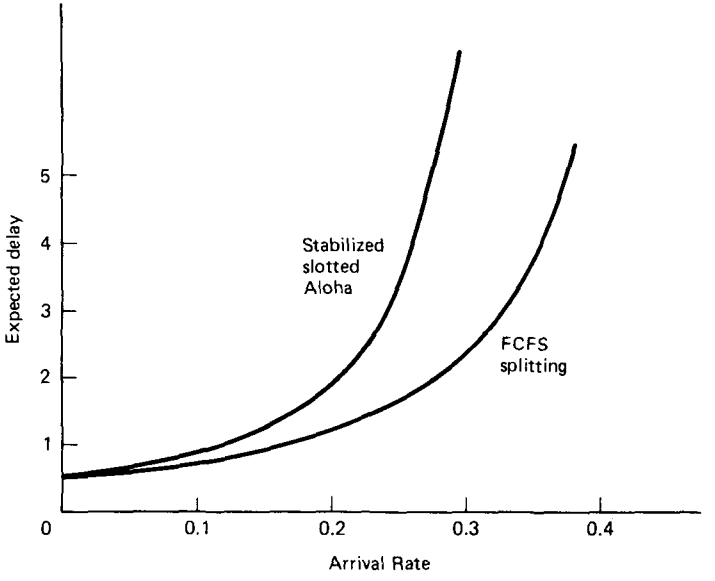


Figure 4.14 Comparison of expected delay for stabilized slotted Aloha and the FCFS splitting algorithm as a function of arrival rate. One becomes unbounded as the arrival rate approaches $1/e$, and the other as the arrival rate approaches 0.4871.

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

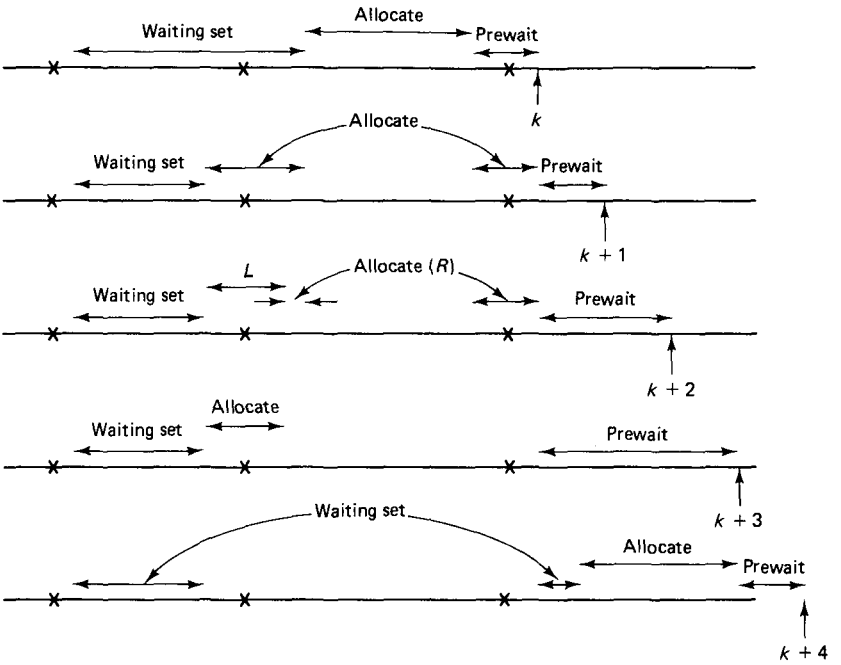


Figure 4.15 Last-come first-serve (LCFS) splitting algorithm. Arrivals wait for the beginning of a new CRP, but then are allocted in last-come first-serve fashion. Allocation sets and waiting sets can consist of more than one interval.

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 [HlG81] 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

$$\beta = \frac{\tau C}{L} \tag{4.33}$$

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 [KlT75] 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

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

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

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

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

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

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

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

where

$$g(n) = \lambda\beta + q_r n \tag{4.38}$$

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

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

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

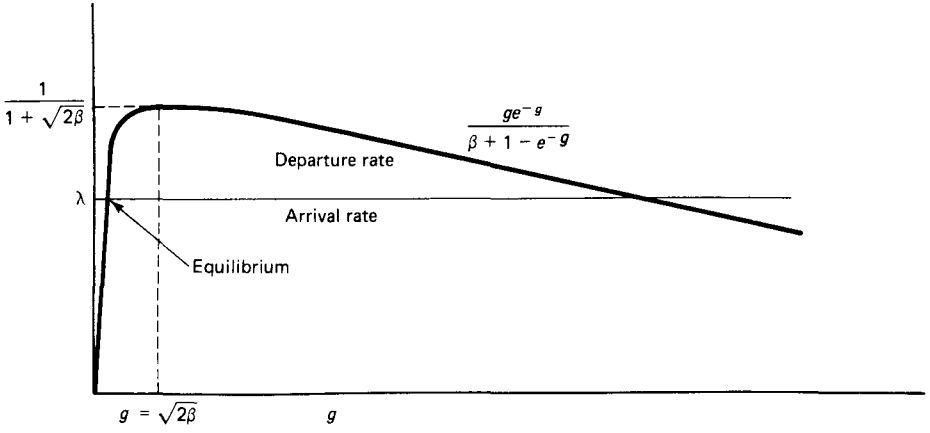


Figure 4.16 Departure rate, in packets per unit time, for CSMA slotted Aloha as a function of the attempted transmission rate g in packets per idle slot. If β, the duration of an idle slot as a fraction of a data slot, is small, the maximum departure rate is $1/(1 + \sqrt{2\beta})$.

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

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

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

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

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

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

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

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

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

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

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

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

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,

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

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

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

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

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

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

$$G(n) = \lambda + nx \tag{4.49}$$

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

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

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,

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

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

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

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

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

$$g \approx \sqrt{\frac{2\beta}{T - 1}} \tag{4.54}$$

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

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,

$$T \approx (1 - x)^2(\beta + T) + 4x(1 - x)(1 + \beta) + x^2(1 + \beta + T) \tag{4.56}$$

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

$$x = \sqrt{\beta + \beta^2} - \beta \tag{4.57}$$

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

$$\lambda_{\max} \approx \frac{1}{1 + \sqrt{2\beta}} \tag{4.58}$$

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

$$S = \frac{1}{1 + v/S_r} \tag{4.59}$$

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

$$S = \frac{1}{1 + v(1/S_R - 1)} \tag{4.60}$$

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

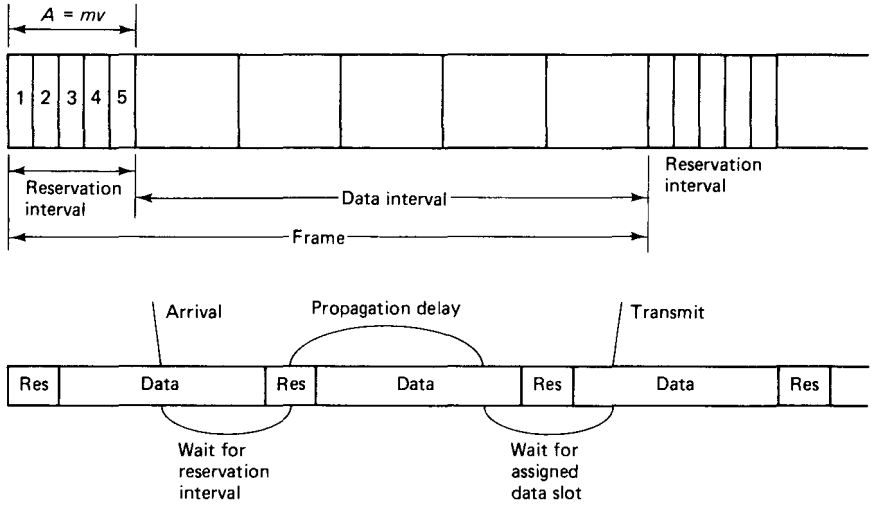


Figure 4.17 Satellite reservation system, using TDM to make reservations. Arrivals in one frame are transmitted in the second following frame.

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

$$E\{W_i\} = E\{R_i\} + \frac{E\{N_i\}}{\mu} + 2A \tag{4.61}$$

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

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

This analysis allows the data packets to have a general length distribution with mean square $\overline{X^2}$ (which of course requires the reservations to contain length information), but we have normalized time to satisfy $\overline{X} = 1/\mu = 1$; 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

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

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

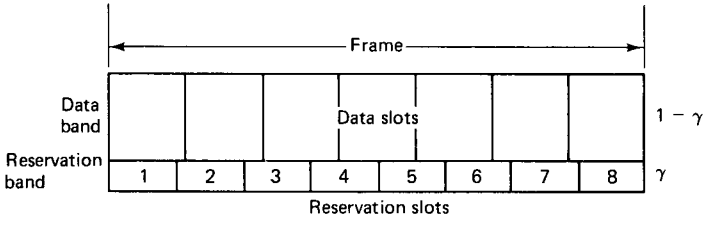


Figure 4.18 Reservation system with separate frequency band for reservations. Reservations are made by TDM in the reservation band.

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;

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

The expected number of arrivals between state transitions is λ times this interval, so the drift in state n is $\lambda E\{\text{interval}\} - P_{succ}$. 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

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

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

$$\lambda < \frac{1}{1 + 3.31\beta} \tag{4.66}$$

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 $\overline{X^2}$, is

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

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

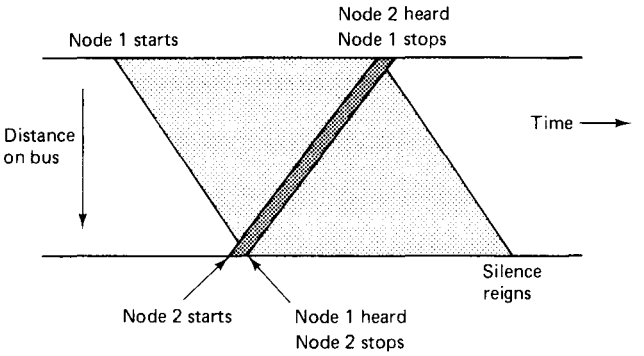


Figure 4.20 Collision detection. Node 2 starts to transmit almost β units after node 1; node 2 stops almost immediately, but node 1 continues for almost another β units before stopping.

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

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

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

$$S > \frac{1}{1 + 6.2\beta} \tag{4.69}$$

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 011111 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.

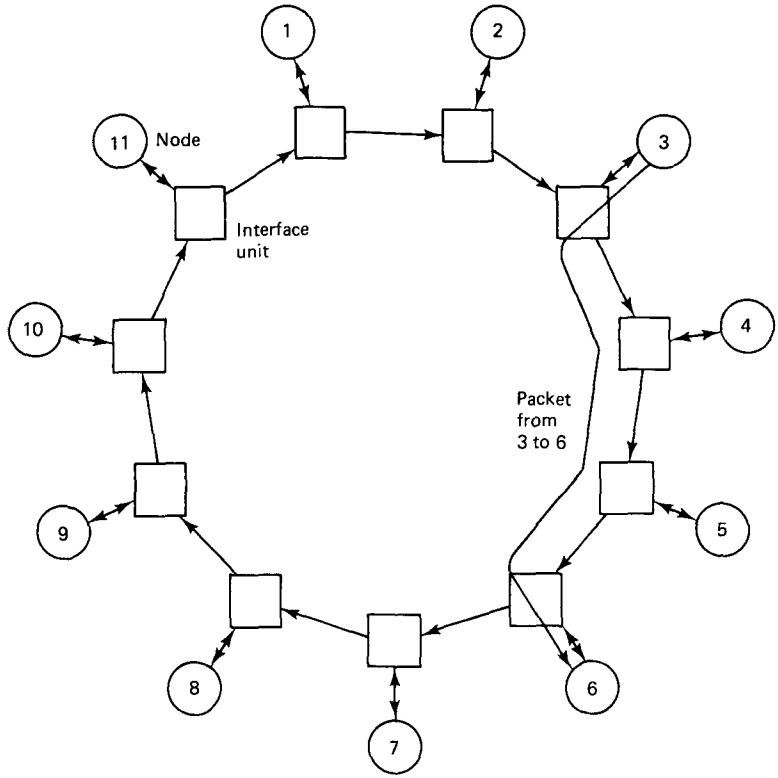


Figure 4.21 Ring network. Travel of data around the ring is unidirectional. Each node either relays the received bit stream to the next node with a one bit delay or transmits its own packet, discarding the incoming bit stream.

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

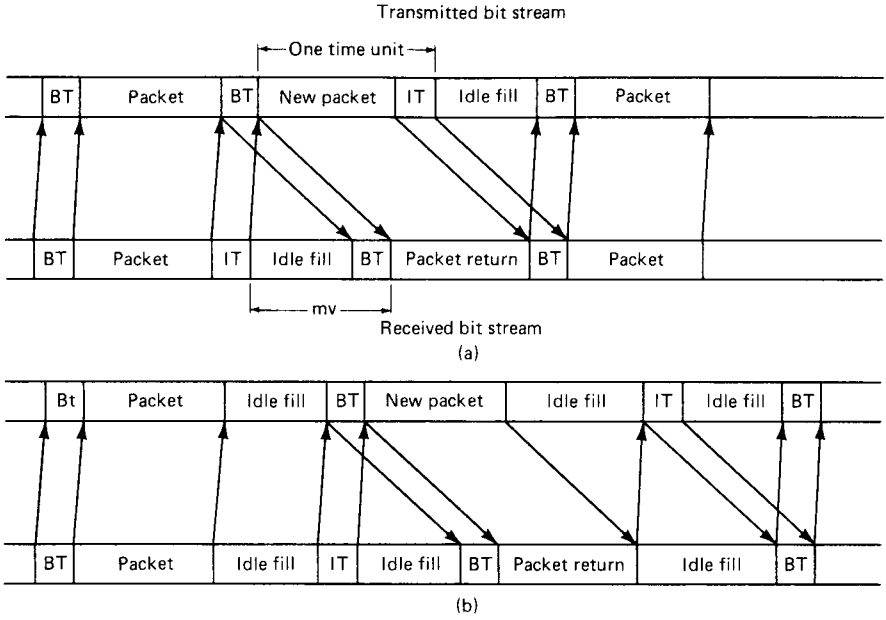


Figure 4.22 Transmitted and received bit stream at one ring interface unit of a token ring network. The interface unit transmits what is received with a one bit delay until seeing an idle token (IT). It converts the idle token into a busy token (BT) and then transmits its own packet. In part (a), this 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

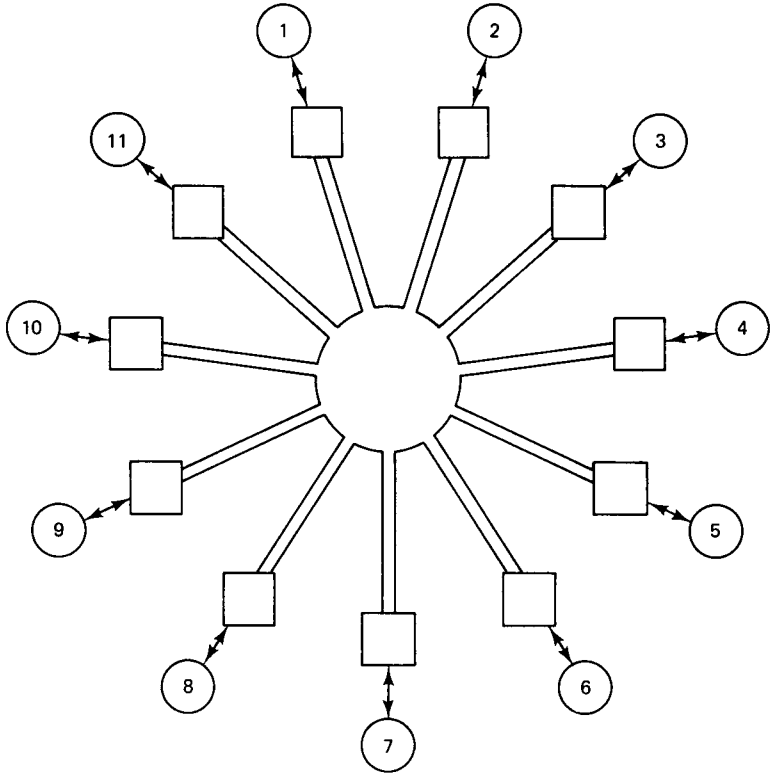


Figure 4.23 Ring network in a star configuration. Nodes can be bypassed or added from the central site.

around the ring. Let $\overline{X} = 1$ 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

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

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

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

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

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

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

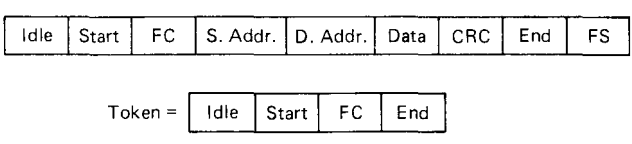


Figure 4.24 Frame format for FDDI.

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 $0, 1, \ldots, m-1$ 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 $\alpha_0 \ldots \alpha_{m-1}$ are allocated in such a way that $\alpha_0 + \alpha_1 + \ldots + \alpha_{m-1} \leq \tau$. To describe and analyze the algorithm precisely, let $t_0, t_1, \ldots, t_{m-1}$ be the times at which the token reaches nodes 0 to $m-1$ for some given cycle. Similarly, let $t_m \ldots t_{2m-1}$ 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 mod 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 mod 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

$$t_{i+1} \leq t_{i-m} + \tau + \alpha_i. \qquad \text{for } t_i - t_{i-m} < \tau;\ i \geq 0$$

$$t_{i+1} \leq t_i + \alpha_i, \qquad \text{for } t_i - t_{i-m} \geq \tau;\ i \geq 0$$

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

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

Note that equality holds if node (i mod 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

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

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

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

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

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

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

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

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 $t'_{j+m}, t'_{j+2m}, \ldots, t'_{j+(m+1)m}$. 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 $[m/(m + 1)]\tau'$ 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

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

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

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

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

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

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

$$t'_i \leq \tau + \left\lfloor \frac{i}{m+1} \right\rfloor \tau'$$

$$t_i \leq \sum_{j=0}^{i-1} \alpha_j + (\tau + \tau) - T \left\lfloor \frac{i}{m+1} \right\rfloor \tag{4.80}$$

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

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

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

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

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 $\limsup_{k \to \infty} (t_{mk}/k) \leq \tau m/(m+1) + T/(m+1)$. 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.

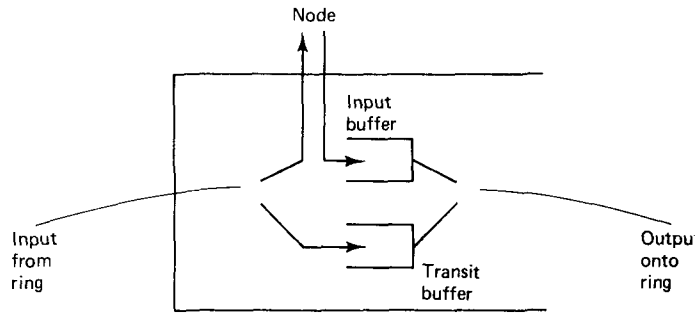


Figure 4.25 Interface to a register insertion ring. The output onto the ring comes from either the input buffer or the transit buffer. The input from the ring goes to the transit buffer or directly to the node, if addressed there.

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 $\lambda \to 0$ 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,

$$v = \frac{\tau C}{3L} + \frac{k}{L} + \delta \tag{4.83}$$

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

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

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 [KlS80]. 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.

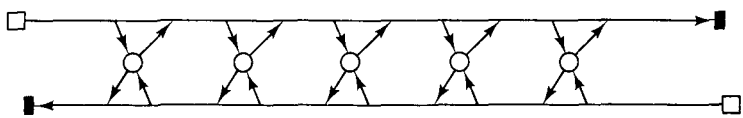


Figure 4.26 Bus structure for IEEE 802.6 dual bus.

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 cleanness 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

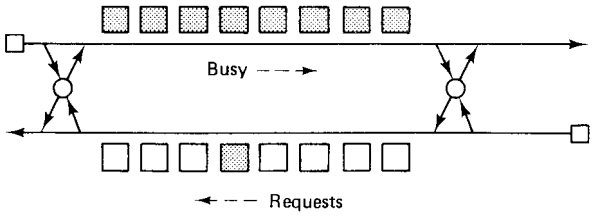


Figure 4.27 The upstream node on the left has a large file of data and is filling each downstream slot. The downstream node then receives a large file to send. The first frame is in the virtual queue and the request bit is traveling upstream.

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.

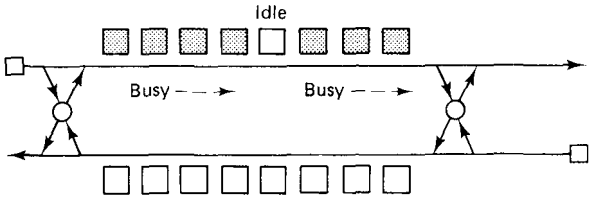


Figure 4.28 Continuation of Fig. 4.27 in which the upstream node allows an idle slot to propagate downstream after receiving a request from the downstream node.

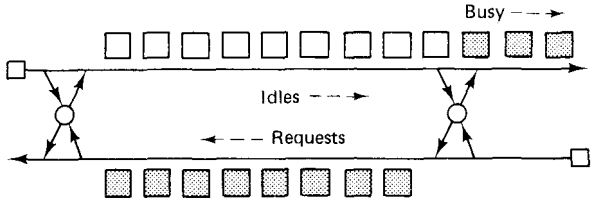


Figure 4.29 Illustration of situation in which a downstream node is sending a frame in every slot just before an upstream node becomes active. Note that the upstream bus is full of slots carrying request bits.

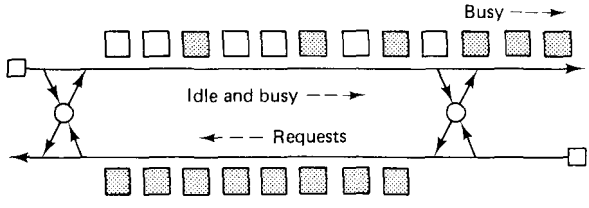


Figure 4.30 Continuation of Fig. 4.29 showing the scenario as the first busy slot arrives at the downstream node. Note that the first busy slot between the two nodes is followed by one idle slot, the second by two idle slots, and so forth.

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,

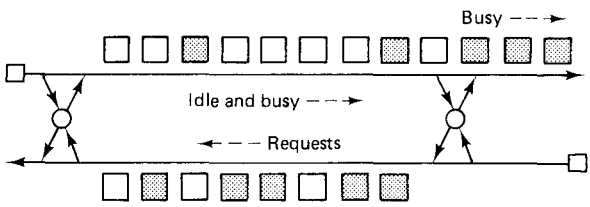


Figure 4.31 Continuation of Fig. 4.29 showing the scenario as the first nonrequest slot arrives at the upstream node. Note that successive nonrequest (busy) slots are followed by increasing numbers of request (idle) slots.

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

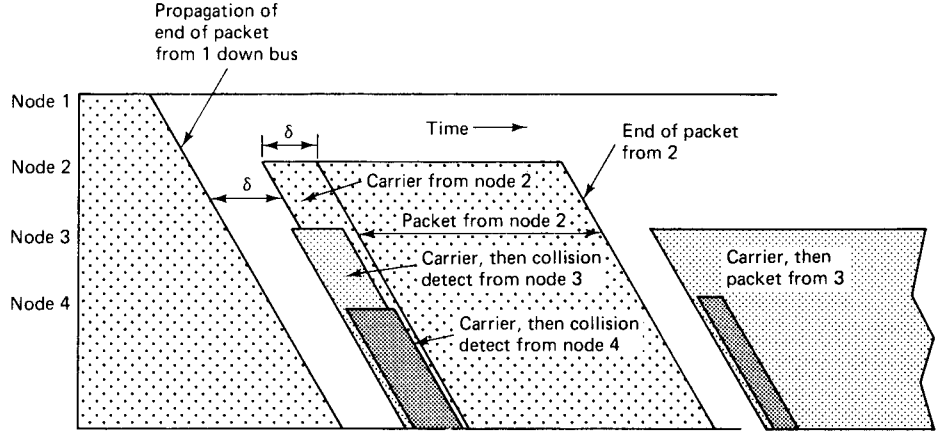


Figure 4.32 Implicit tokens on a unidirectional bus. Each node with traffic transmits carrier on hearing silence, then defers to upstream nodes on hearing carrier.

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

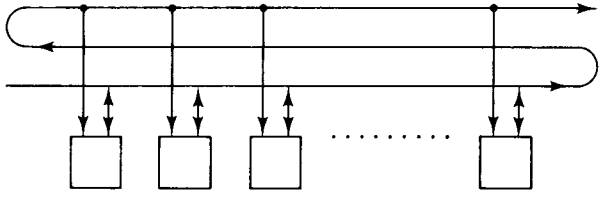


Figure 4.33 Bus structure for Expressnet. Each node transmits on the lower portion of the bus and listens on both lower and upper portions. Contention is resolved on the lower portion with the leftmost node taking priority. Reception and detection of the end of a cycle takes place from the upper portion.

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

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

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

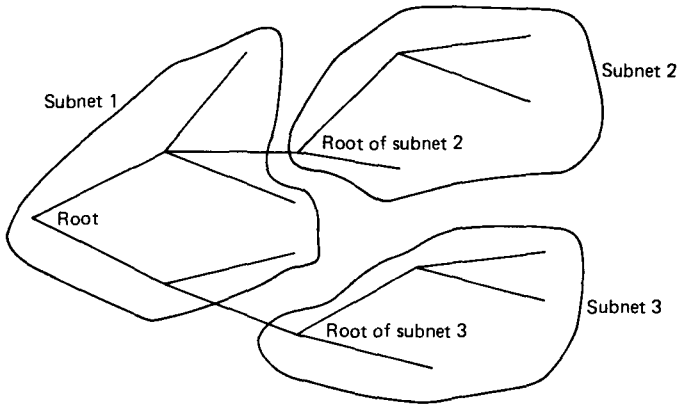


Figure 4.34 A CATV network divided into subnets in the Homenet strategy. Each subnet has one incoming and one outgoing frequency band and uses CSMA/CD to resolve collisions with a subnet.

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 $n_1, \ldots, n_k$ 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 $y_1, \ldots, y_k$.

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.

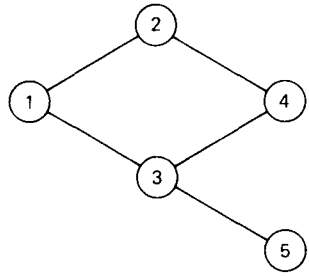


Figure 4.36 Packet radio network: each edge indicates that two nodes at the ends of the edge can hear each other's transmission.

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 $x_1, \ldots, x_J$ are the CFVs corresponding to a collection of J collision-free sets, the vector $f = \left(\sum_j x_j\right)/J$ 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

$$f = \sum_j \alpha_j x_j \tag{4.86}$$

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 $x_1, \ldots, x_J$. 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,

$$Q_i = \sum_j q_{ij} \tag{4.87}$$

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

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

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

$$f_{ij} = q_{ij}\, p_{ij} \tag{4.89}$$

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

$$q_{ij}^1 = \frac{f_{ij}}{p_{ij}^0} \tag{4.90}$$

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,

$$f = q(1 - nq)^n \tag{4.91}$$

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 [KlT75] 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 $p_0, p_1, \ldots, p_n$.

(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

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

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

4.3 **(a)** Redraw Fig. 4.4 for the case in which $q_r = 1/m$ and $q_a = 1/me$.

(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 $q_a = 1/me$) 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 $\overline{n}$ be the expected value of n_k over all k. Note that $\overline{n}$ will depend on the particular way in which collisions are resolved, but we regard $\overline{n}$ as given here (see [HIG81]).

(a) Find the expected number of *accepted* arrivals per slot, $\overline{N}_a$, as a function of $\overline{n}, m$, and q_a, where m is the number of nodes and q_a is the arrival rate per node.

(b) Find the expected departure rate per slot, $\overline{P}_{succ}$, as a function of $\overline{n}, m$, and q_a. *Hint:* How is $\overline{N}_a$ related to $\overline{P}_{succ}$? Recall that both are averages over time.

(c) Find the expected number of packets in the system, $\overline{N}_{sys}$, immediately after the beginning of a slot (the number in the system is the backlog plus the accepted new arrivals).

(d) 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.

(e) Suppose that the strategy for resolving collisions is now modified and the expected backlog $\overline{n}$ is reduced to $\overline{n}' < \overline{n}$. Show that $\overline{N}_a$ increases, $\overline{P}_{succ}$ increases, $\overline{N}_{sys}$ 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.

(a) 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?

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

(c) 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 $\overline{n}$ in Problem 4.4, this is the same as the value of T found there.)

(d) 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

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

(b) 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 $q_a < 1/m$).

(c) 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 $\overline{X}$ between successful transmissions and the second moment $\overline{X^2}$ 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 $\overline{v}$ until the completion of such a successful transmission and its second moment $\overline{v^2}$.

(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

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

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 $\overline{n}$, the expected number of packets successfully transmitted in a CRP of the FCFS splitting algorithm, is given by

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

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

(b) Show that

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

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 $\overline{X} = 1$.

(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 $\overline{Y} \leq 2\overline{X}$. 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 $p(\tau \mid n) = (\lambda + xn)e^{-(\lambda+xn)\tau}$.

(a) Find the unconditional probability density $p(\tau)$.

(b) Find the a posteriori probability $P\{n, b \mid \tau\}$ 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 $P\{n, a \mid \tau\}$ 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,

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

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,

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

(c) Evaluate $\overline{R}$ and $\overline{y}$ 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.

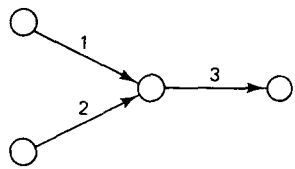


Figure 4.37

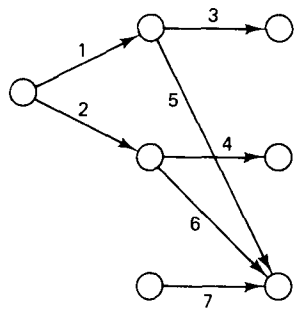


Figure 4.38

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 $\alpha_0, \ldots, \alpha_{m-1}$. 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 arbitarily large backlogs. Assume that $T = \alpha_0 + \alpha_1 + \cdots + \alpha_{m-1}$ 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.