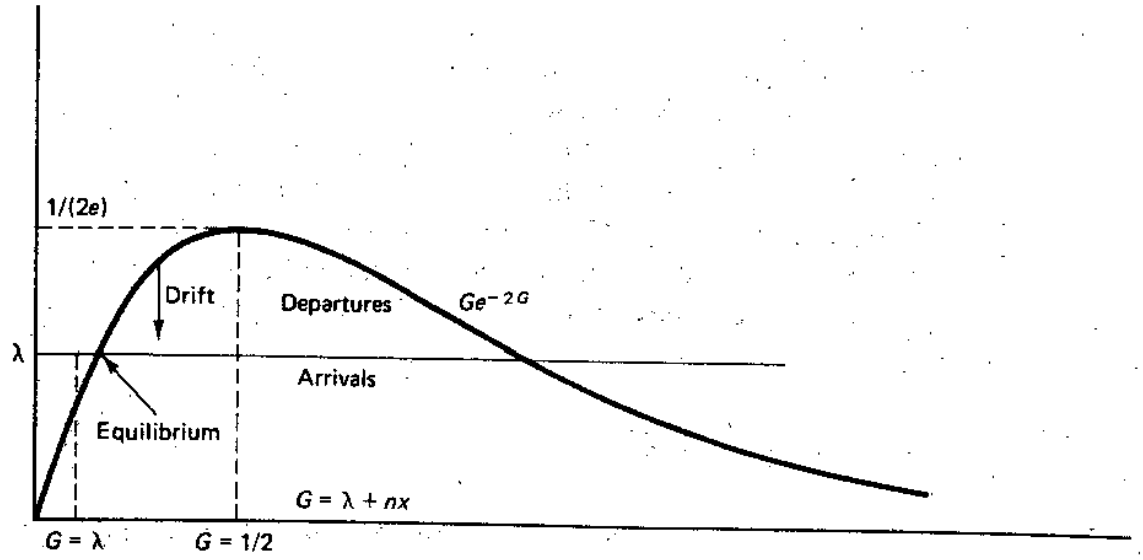


Figure 4.8 Pure Aloha as a function of the attempted transmission rate G. Successful departures leave the system at a rate Ge^{-2G}, and arrivals occur at a rate λ, leading to a hypothesized equilibrium at the point shown.

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.

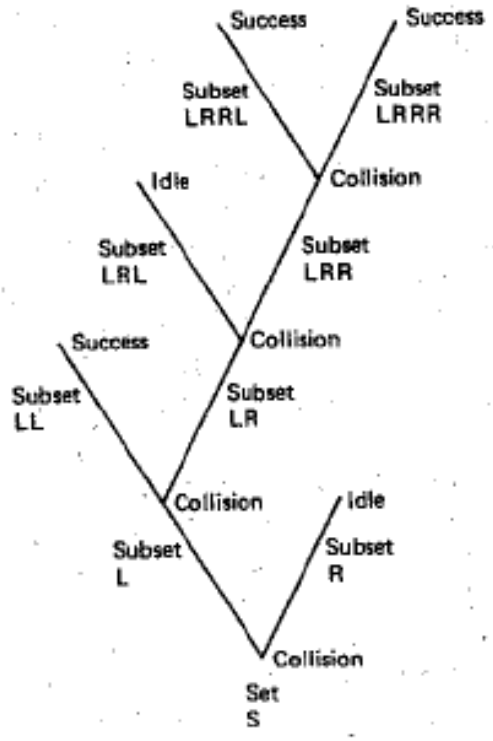
There are various ways in which the nodes involved in a collision could choose whether or not to transmit in successive slots. Each node could simply flip an unbiased coin for each choice. Alternatively (in a way to be described precisely later) each node could use the arrival time of its collided packet. 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.

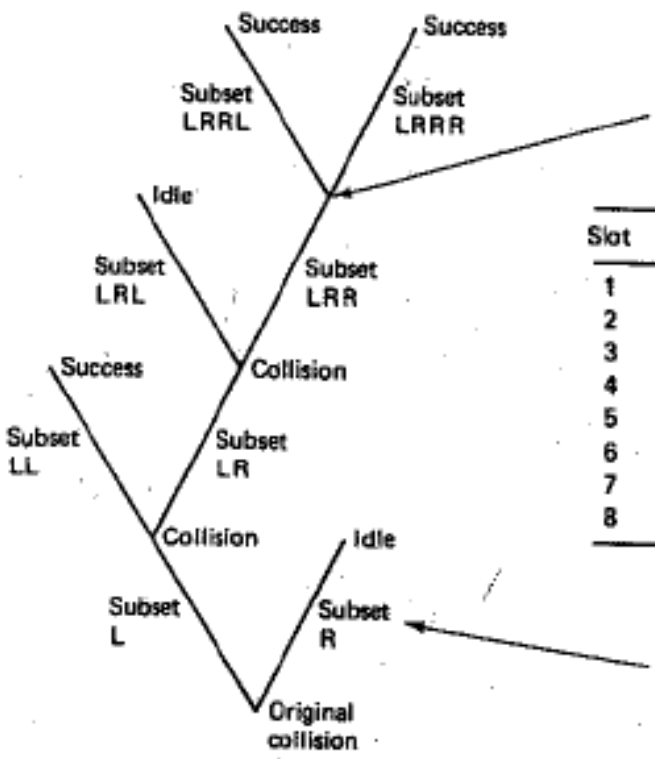
Improvements to the tree algorithm. First consider the situation in Fig. 4.10. Here, in slots 4 and 5, a collision is followed by an idle slot; this means that all the packets involved in the collision were assigned to the second subset. The tree algorithm would simply transmit this second subset, generating 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].