Exercise 5 — LANs
Sample Solution

1. As I hinted in class, this problem is pretty open-ended. There is certainly no single, correct solution. Rather, what I was hoping was that you would find some creative ways to exploit certain properties of the acknowledgements used in the data link protocols we have considered.

   One of the key properties of acks I had in mind is the fact that in most data link protocols they are cummulative. That is, an ack does not just verify that a given packet has arrived but that all packets up to and including the acknowledged packet have arrived. In a non-broadcast setting, this fact implies that if a single ack gets lost, the following acks will takes its place. In a broadcast setting you might consider taking advantage of this to reduce the number of acks sent by not having every station acknowledge every packet. For example, if there are N stations involved in communicating and you number them 0 through N-1, then you might have station j only attempt to acknowledge packets whose sequence number is equal to j mod N. Basically, the receivers will take turns sending acks. As long as everything works right, each packet will generate just one ack.

   This by itself isn't enough. If a transmission is garbled so that everyone fails to receive it, it will be retransmitted promptly. If, however, just a few stations miss the packet and the station whose turn it is to send an ack does not miss, retransmission may be quite delayed.

   That is where the other issue I wanted you to think about comes in. Namely, the roles of positive and negative acknowledgements. In a broadcast setting, machines will hear each other's acks. So, even if a station misses a packet, it may receive the packet's ack. This will be enough to let it know that a retransmission would help. In such situations, the use of Naks might improve efficiency. The important issue, however, is to recognize that acks are ultimately responsible for reliability even if naks are used to improve efficiency.

   Since there are many legitimate ways one might have worked out the details of such a protocol, I won't go into additional depth on any particular approach here. If you are curious, such a scheme was described in detail in a paper entitled "Reliable Broadcast Protocols" by Chang and Maxemchuk that appeared in the ACM's Transactions on Computer Systems in August of 1984.

2. (a) The IEEE 802.5 standard included provisions for making reservations based on priorities. As the text explains, a requesting station would set its priority in a message as it passed by and then this information would be used to determine the priority of the new token when it was generated. This implies that the new token could not be generated until at least the header of the packet sent returned to the sender. Therefore, choosing to delay the release of a free token until a packet's header returned simplified the design of the priority system. Given the speeds of the networks in question, this was enough of a reason not to base the 802.5 protocol on early release of the token.

   (b) Interestingly, if one ignores the priority scheme and the token ring's error recovery mechanisms, it would be feasible to construct a ring in which different stations took different approachs to releasing a new free token. Interfaces based on releasing the token only after the header of the packet sent makes a complete round trip have to be designed to accept a token that immediately follows a packet. Such a "train" will occur on all but large rings since on smaller rings the two token release schemes are effectively identical. Interfaces based on early release, on the other hand could easily be designed to cope with situations where a packet train was followed by idle time rather than a free token.

3. Stallings' analysis is based on the assumption that the token is not released until the header has returned. This is why his final description of S is broken down into two cases. The $a < 1$ case applies when packets fill the ring and therefore the token can be released as soon as transmission is complete since the header is certain to have returned by then. The $a > 1$ case accounts for the extra delay waiting for the header before freeing a token when transmission takes less time than propagation.

   The analysis of the other approach, early token release, is simpler because there is only one case. This case is identical to the analysis of the late token release scheme when $a < 1$. Namely, it takes unit time to send a packet after which the token is immediately released and moves to the next station in expected time a/N. So, the formula for S in this case would be:

$$\frac{1}{1 + a/N}$$